

AIM OF THE EXPERIMENT:

- Write a program to find out if an element is a member of the list.
- Write a program extract kth element from list.
- Write a program to print all the elements of the list.
- Write a Prolog program to implement `sumlist (List,Sum)` so that Sum is the sum of a given list of numbers List.
- Write a program in Prolog to find out whether a number is greater than, equal or lesser than the other number

THEORY:

Recursion is a technique in which one predicate uses itself (may be with some other predicates) to find the truth value.

Let us understand this definition with the help of an example -

- `is_digesting(X,Y) :- just_ate(X,Y).`
- `is_digesting(X,Y) :- just_ate(X,Z), is_digesting(Z,Y).`

So this predicate is recursive in nature. Suppose we say that `just_ate(deer, grass)`, it means `is_digesting(deer, grass)` is true. Now if we say `is_digesting(tiger, grass)`, this will be true if `is_digesting(tiger, grass) :- just_ate(tiger, deer), is_digesting(deer, grass)`, then the statement `is_digesting(tiger, grass)` is also true.

There may be some other examples also, so let us see one family example. So if we want to express the predecessor logic, that can be expressed using the following diagram -

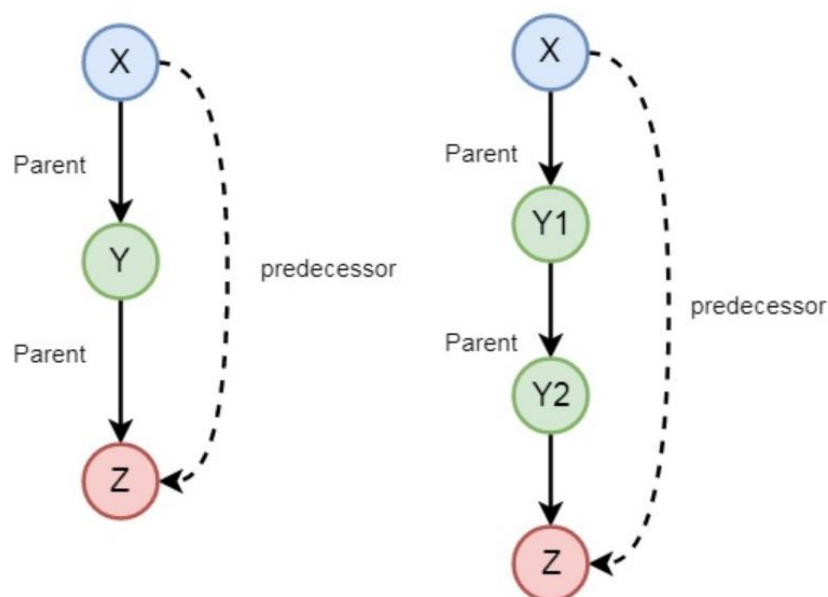


Fig 2.1 Recursion explanation using diagram

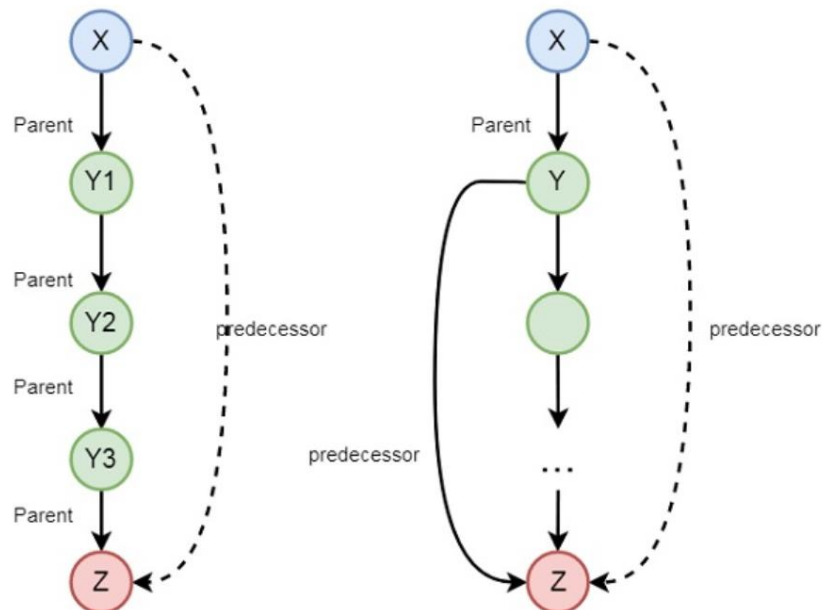


Fig 2.2 Recursion explanation using diagram

So we can understand the predecessor relationship is recursive. We can express this relationship using the following syntax -

- `predecessor(X, Z) :- parent(X, Z).`
- `predecessor(X, Z) :- parent(X, Y),predecessor(Y, Z).`

Concept of list: List is a data structure that can be used in different cases for non-numeric programming. Lists are used to store the atoms as a collection. List consists of any number of items, for example, red, green, blue, white, dark. It will be represented as, [red, green, blue, white, dark]. The list of elements will be enclosed with square brackets.

A list can be either empty or non-empty. In the first case, the list is simply written as a Prolog atom, []. In the second case, the list consists of two things as given below -

- The first item, called the head of the list;
- The remaining part of the list, called the tail.
- Suppose we have a list like: [red, green, blue, white, dark]. Here the head is red and tail is [green, blue, white, dark]. So the tail is another list.

Operations	Definition
Membership Checking	During this operation, we can verify whether a given element is member of specified list or not?
Length Calculation	With this operation, we can find the length of a list.
Concatenation	Concatenation is an operation which is used to join/add two lists.
Delete Items	This operation removes the specified element from a list.
Append Items	Append operation adds one list into another (as an item).
Insert Items	This operation inserts a given item into a list.

Fig 2.3 List operations in Prolog

SOURCE CODE:

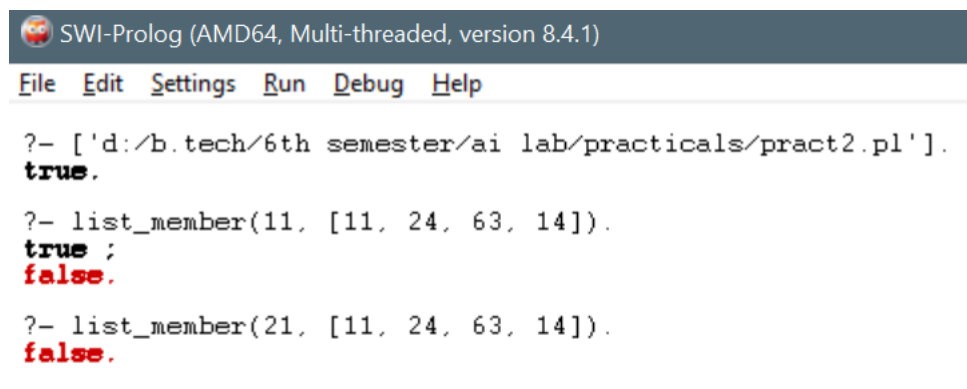
- Write a program to find out if an element is a member of the list.



```
pract2.pl
File Edit Browse Compile Prolog Pce Help
pract2.pl
list_member(X, [X|_]).
list_member(X, [_|TAIL]) :- list_member(X, TAIL).
```

Fig 2.4 Program to find if an element is a member of the list

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

?- ['d:/b.tech/6th semester/ai lab/practicals/pract2.pl'].
true.

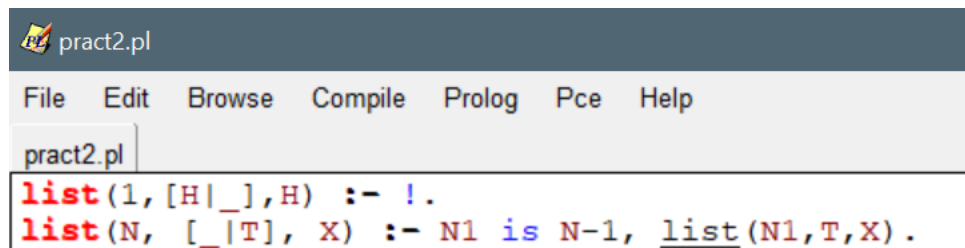
?- list_member(11, [11, 24, 63, 14]).
true ;
false.

?- list_member(21, [11, 24, 63, 14]).
false.
```

Fig 2.5 Output

SOURCE CODE:

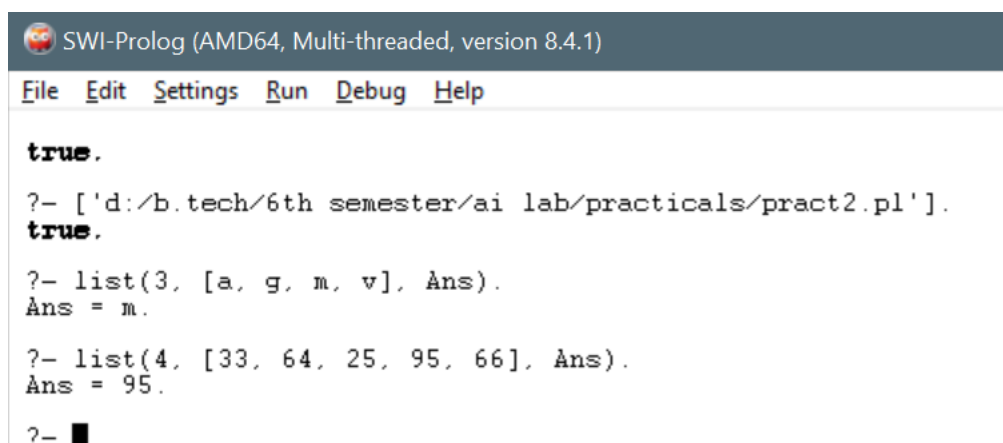
- b. Write a program extract kth element from list



```
pract2.pl
File Edit Browse Compile Prolog Pce Help
pract2.pl
list(1, [H|_], H) :- !.
list(N, [_|T], X) :- N1 is N-1, list(N1, T, X).
```

Fig 2.6 Program to extract kth element from the list

OUTPUT:



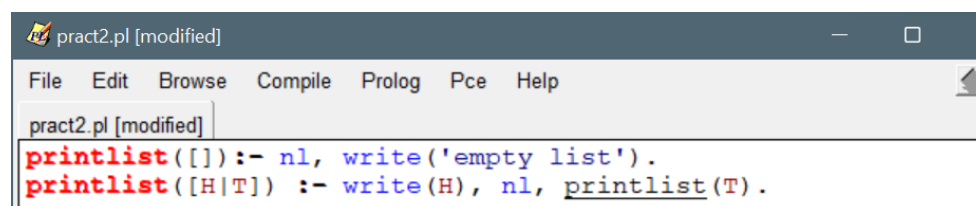
```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

true.
?- ['d:/b.tech/6th semester/ai lab/practicals/pract2.pl'].
true.
?- list(3, [a, g, m, v], Ans).
Ans = m.
?- list(4, [33, 64, 25, 95, 66], Ans).
Ans = 95.
?-
```

Fig 2.7 Output

SOURCE CODE:

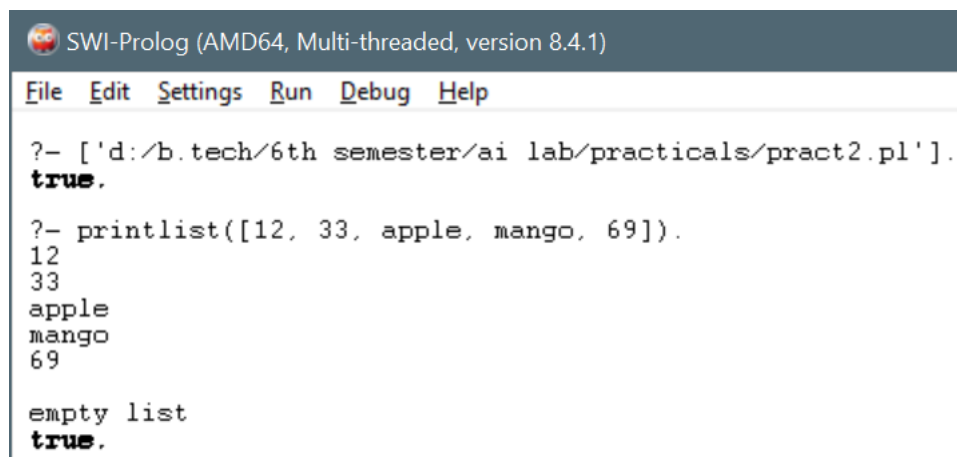
- c. Write a program to print all the elements of the list.



```
pract2.pl [modified]
File Edit Browse Compile Prolog Pce Help
pract2.pl [modified]
printlist([]) :- nl, write('empty list').
printlist([H|T]) :- write(H), nl, printlist(T).
```

Fig 2.8 Program to print all the elements in list

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

?- ['d:/b.tech/6th semester/ai lab/practicals/pract2.pl'].
true.

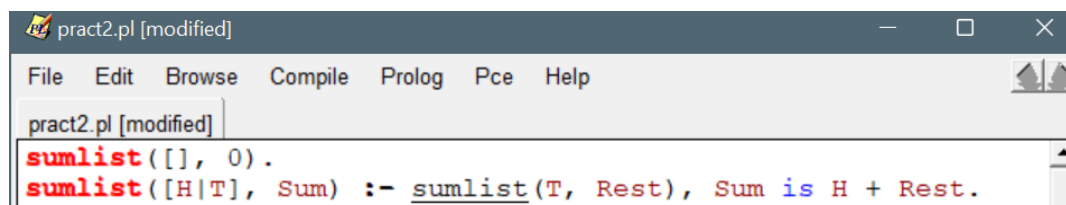
?- printlist([12, 33, apple, mango, 69]).
12
33
apple
mango
69

empty list
true.
```

Fig 2.9 Output

SOURCE CODE:

- d. Write a Prolog program to implement `sumlist (List,Sum)` so that `Sum` is the sum of a given list of numbers `List`.

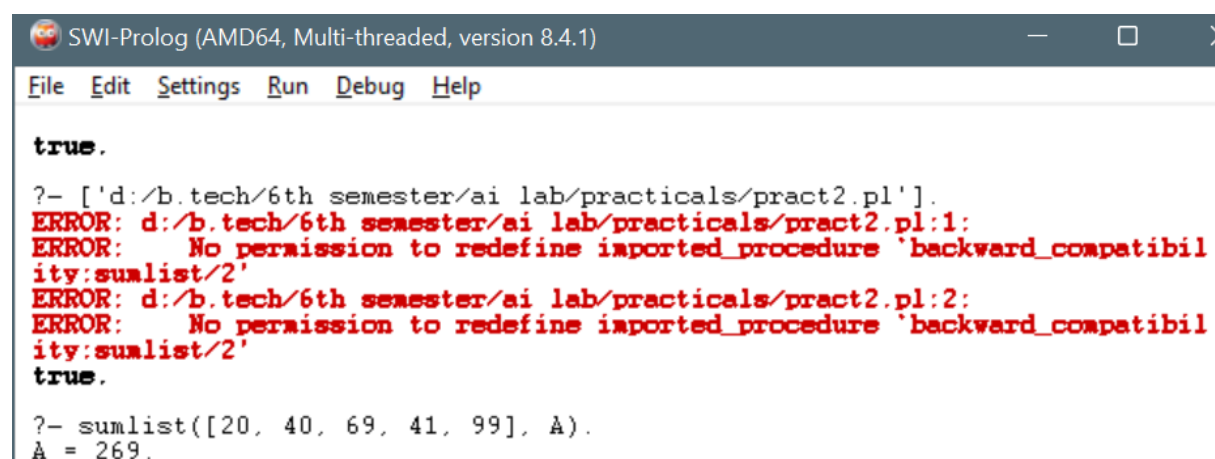


```
pract2.pl [modified]
File Edit Browse Compile Prolog Pce Help

pract2.pl [modified]
sumlist([], 0).
sumlist([H|T], Sum) :- sumlist(T, Rest), Sum is H + Rest.
```

Fig 2.10 Program to add all the elements in list

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

true.

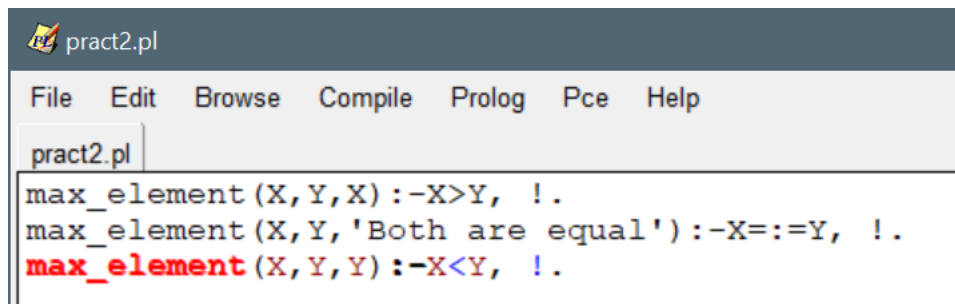
?- ['d:/b.tech/6th semester/ai lab/practicals/pract2.pl'].
ERROR: d:/b.tech/6th semester/ai lab/practicals/pract2.pl:1:
ERROR:    No permission to redefine imported_procedure 'backward_compatibil
ity:sumlist/2'
ERROR: d:/b.tech/6th semester/ai lab/practicals/pract2.pl:2:
ERROR:    No permission to redefine imported_procedure 'backward_compatibil
ity:sumlist/2'
true.

?- sumlist([20, 40, 69, 41, 99], A).
A = 269.
```

Fig 2.11 Output

SOURCE CODE:

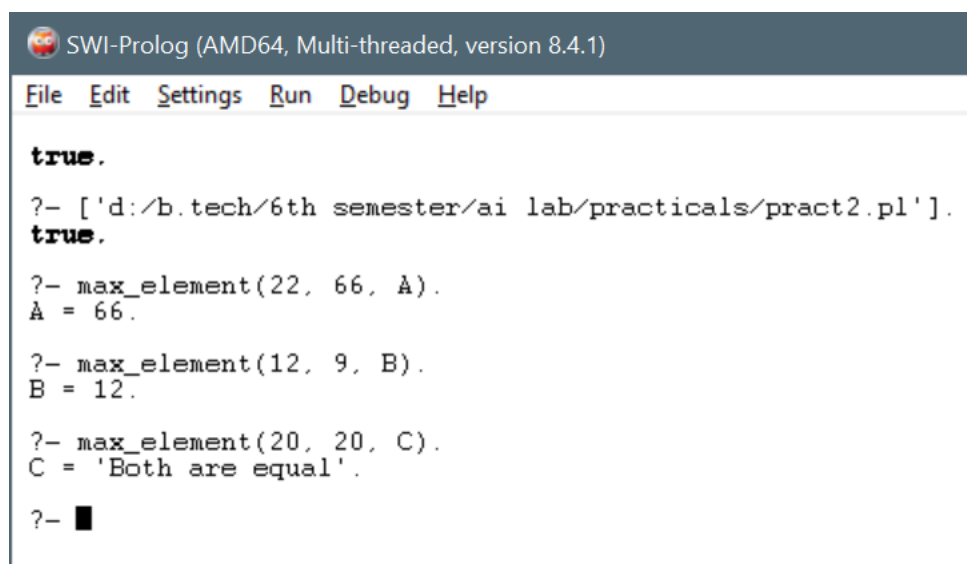
- e. Write a program in Prolog to find out whether a number is greater than, equal or lesser than the other number



```
pract2.pl
File Edit Browse Compile Prolog Pce Help
pract2.pl
max_element(X,Y,X):-X>Y, !.
max_element(X,Y,'Both are equal'):-X==Y, !.
max_element(X,Y,Y):-X<Y, !.
```

Fig 2.12 Program to find whether a number is greater than, less than, or equal to the other number

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

true.
?- ['d:/b.tech/6th semester/ai lab/practicals/pract2.pl'].
true.
?- max_element(22, 66, A).
A = 66.
?- max_element(12, 9, B).
B = 12.
?- max_element(20, 20, C).
C = 'Both are equal'.
?-
```

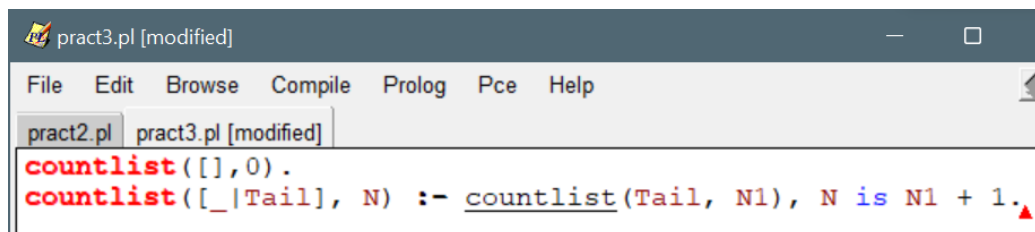
Fig 2.13 Output

AIM OF THE EXPERIMENT:

- Write a Prolog program to implement countlist (List, Count) so that Count is the count of given list of numbers List.
- Write a program in prolog to concatenate one list to the other list.

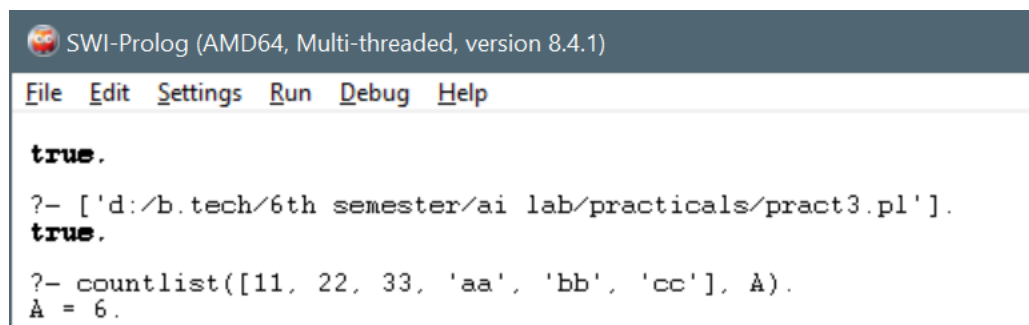
SOURCE CODE:

- Write a Prolog program to implement countlist (List, Count) so that Count is the count of given list of numbers List.



```
pract3.pl [modified]
File Edit Browse Compile Prolog Pce Help
pract2.pl pract3.pl [modified]
countlist([],0).
countlist(_|Tail, N) :- countlist(Tail, N1), N is N1 + 1.
```

Fig 3.1 Program to count number of elements in a List

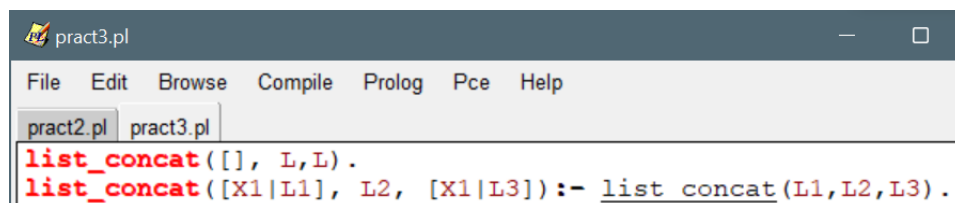
OUTPUT:


```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
true.
?- ['d:/b.tech/6th semester/ai lab/practicals/pract3.pl'].
true.
?- countlist([11, 22, 33, 'aa', 'bb', 'cc'], A).
A = 6.
```

Fig 3.2 Output

SOURCE CODE:

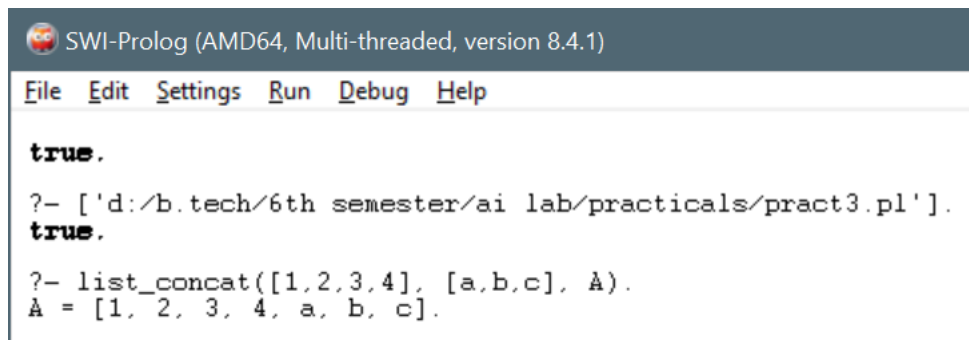
- Write a program in prolog to concatenate one list to the other list.



```
pract3.pl
File Edit Browse Compile Prolog Pce Help
pract2.pl pract3.pl
list_concat([], L,L).
list_concat([X1|L1], L2, [X1|L3]) :- list_concat(L1,L2,L3).
```

Fig 3.3 Program to concatenate two Lists

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

true.
?- ['d:/b.tech/6th semester/ai lab/practicals/pract3.pl'].
true.
?- list_concat([1,2,3,4], [a,b,c], A).
A = [1, 2, 3, 4, a, b, c].
```

Fig 3.4 Output

AIM OF THE EXPERIMENT:

- Write a program in Prolog to find the factorial of a number.
- WAP a program in Prolog to show the working of the following built-in predicates in Prolog

THEORY:

- To find the **factorial** of a number, multiply the number with the factorial value of the previous number. For example, to know the value of 6! multiply 120 (the factorial of 5) by 6, and get 720. For 7!
- Prolog **predicate** is the method to contain the argument and return the boolean values such as true or false. It is a function to operate and return given values, variables, or arguments using a prolog programming language.
- Built-in predicates:** The predicate's definition is provided in advance by the Prolog system, instead of by your own clauses.
- There are three types of built-in predicates as given below -
 - Identifying terms
 - Decomposing structures
 - Collecting all solutions

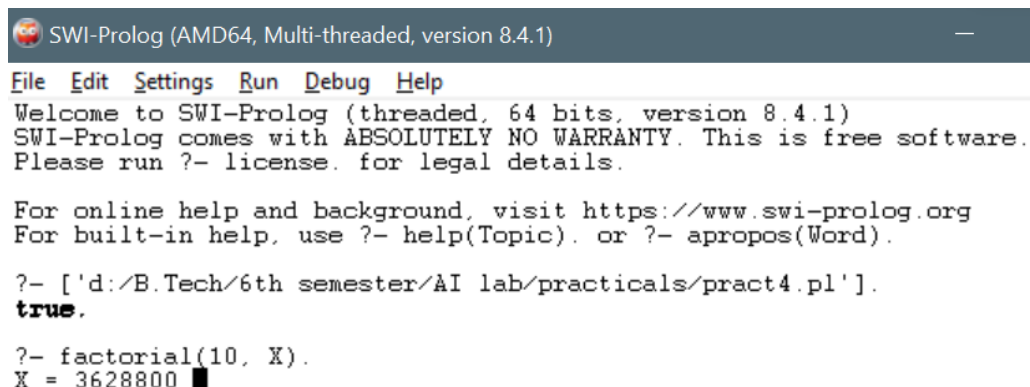
SOURCE CODE:

- Write a program in Prolog to find the factorial of a number.

```
factorial(0, 1).
factorial(N, F) :- N > 0, Prev is N-1, factorial(Prev, R), F is R*N.
```

Fig 4.1 Program to find the factorial of a number

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

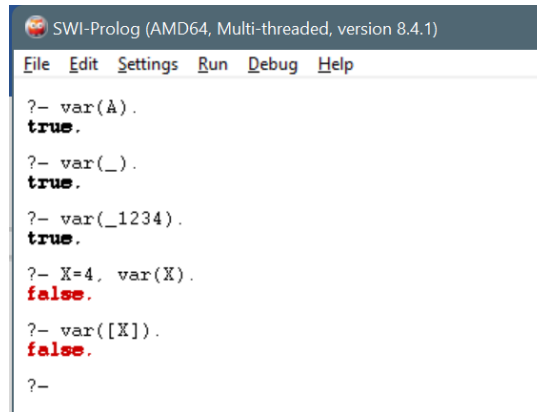
?- ['d:/B.Tech/6th semester/AI lab/practicals/pract4.pl'].
true.

?- factorial(10, X).
X = 3628800
```

Fig 4.2 Output

SOURCE CODE:

- b. WAP a program in Prolog to show the working of the following built-in predicates in Prolog
- i. `var(X)`- When X is not initialized, then, it will show true, otherwise false.

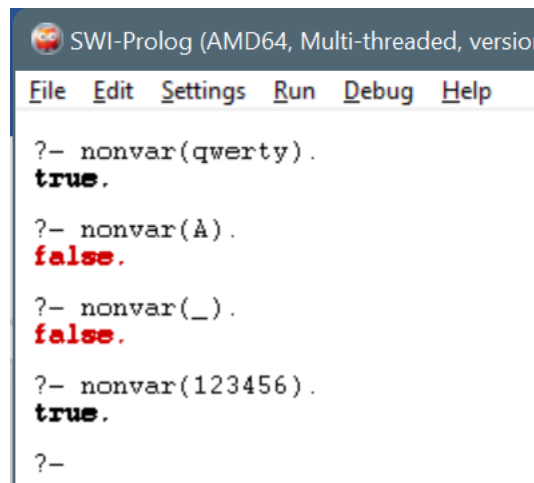


```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

?- var(A).
true.
?- var(_).
true.
?- var(_1234).
true.
?- X=4, var(X).
false.
?- var([X]).
false.
?-
```

Fig 4.3 var - built in predicate in Prolog

- ii. `nonvar(X)`- When X is not initialized, the, it will show false, otherwise true.

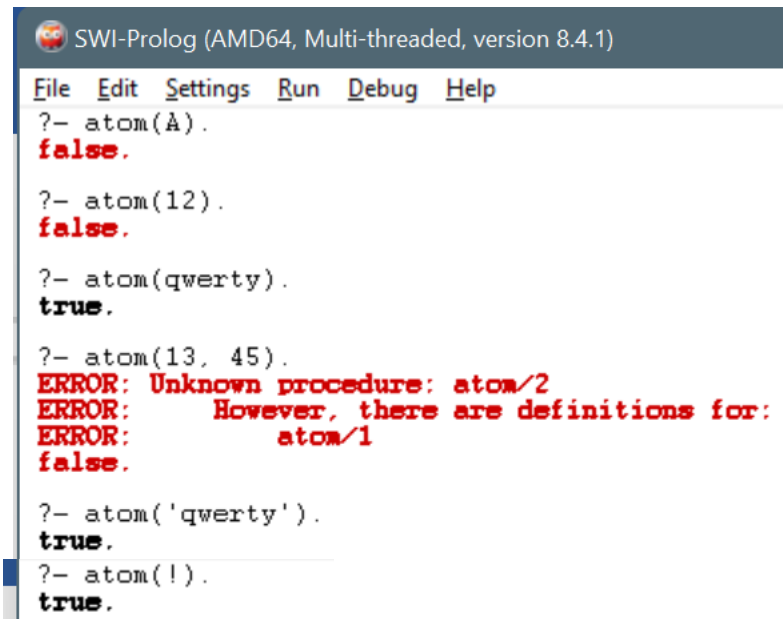


```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

?- nonvar(qwerty).
true.
?- nonvar(A).
false.
?- nonvar(_).
false.
?- nonvar(123456).
true.
?-
```

Fig 4.4 nonvar - built in predicate in Prolog

- iii. `atom(X)`- This will return true, when a non-variable term with 0 argument and a not numeric term is passed as X, otherwise false.



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- atom(A).
false.

?- atom(12).
false.

?- atom(qwerty).
true.

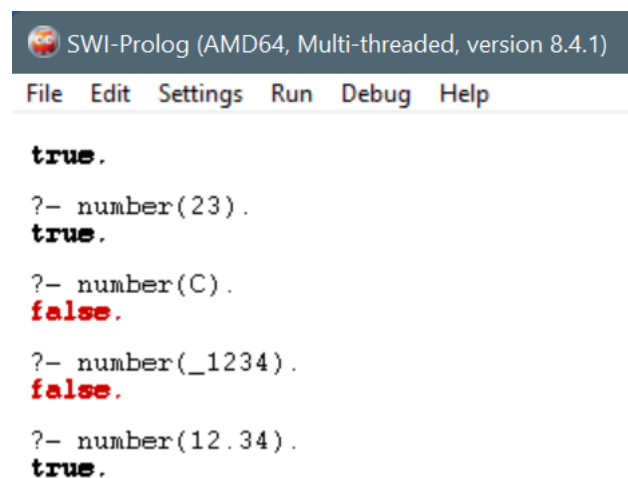
?- atom(13, 45).
ERROR: Unknown procedure: atom/2
ERROR:         However, there are definitions for:
ERROR:         atom/1
false.

?- atom('qwerty').
true.

?- atom(!).
true.
```

Fig 4.5 atom - built in predicate in Prolog

- iv. number(X)- This will return true, X stands for any number, otherwise false.



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

true.

?- number(23).
true.


?- number(C).
false.

?- number(_1234).
false.

?- number(12.34).
true.
```

Fig 4.6 number - built in predicate in Prolog


- v. integer(X)- This will return true, when X is a positive or negative integer value, otherwise false.

 SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

true.
?- integer(16).
true.
?- integer(78.452).
false.
?- integer(-34).
true.
?- integer(C).
false.
?- integer(78/452).
false.

Fig 4.7 integer - built in predicate in Prolog

- vi. float(X)- This will return true, X is a floating-point number, otherwise false.

 SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

?- float(5.8).
true.
?- float(-98.21).
true.
?- float(5).
false.

Fig 4.8 float - built in predicate in Prolog

- vii. atomic(X)- We have atom(X), that is too specific, it returns false for numeric data, the atomic(X) is like atom(X) but it accepts number.

```
?- atomic(russia).  
true.  
?- atomic('russia').  
true.  
?- atomic([q,w,e,r,t]).  
false.  
?- X=ded,atom(X).  
X = ded.
```

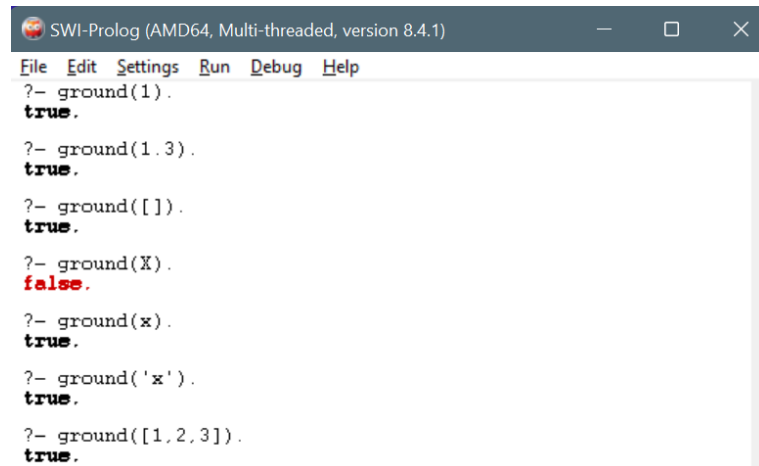
Fig 4.9 atomic - built in predicate in Prolog

- viii. `compound(X)`- If `atomic(X)` fails, then the terms are either one non-instantiated variable (that can be tested with `var(X)`) or a compound term. Compound will be true when we pass some compound structure.

```
?- compound([1,2,3,4,5,6]).  
true.  
  
?- compound([56,43, 'H', a, 3.5, Q]).  
true.
```

Fig 4.10 `compound` - built in predicate in Prolog

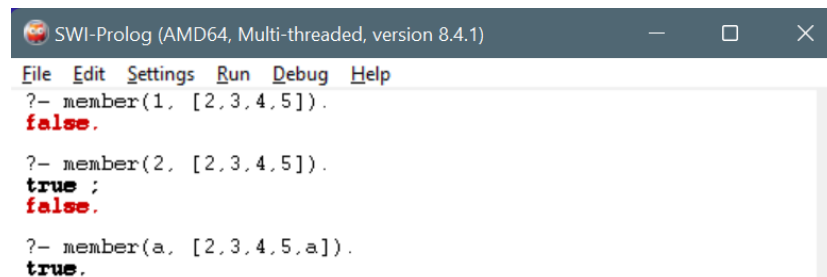
- ix. The `ground(X)` Predicate: This will return true if X does not contain any un-instantiated variables. This also checks inside the compound terms, otherwise returns false.



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)  
File Edit Settings Run Debug Help  
?- ground(1).  
true.  
  
?- ground(1.3).  
true.  
  
?- ground([]).  
true.  
  
?- ground(X).  
false.  
  
?- ground(x).  
true.  
  
?- ground('x').  
true.  
  
?- ground([1,2,3]).  
true.
```

Figure 4.11 `ground` in Prolog

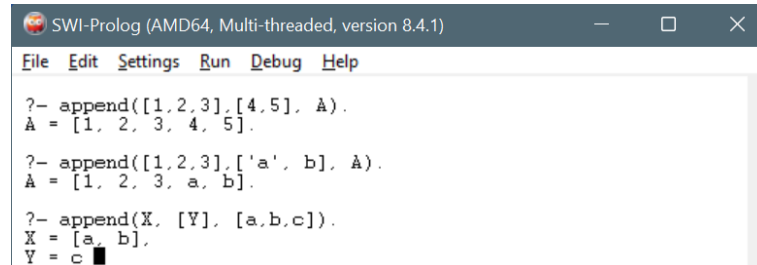
- x. Predicate `member/2`: `member(?Elem, ?List)`.
- True if `Elem` is a member of `List`. The SWI-Prolog definition differs from the classical one. Our definition avoids unpacking each list element twice and provides determinism on the last element. E.g. this is deterministic: `member(X, [One])`.



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)  
File Edit Settings Run Debug Help  
?- member(1, [2,3,4,5]).  
false.  
  
?- member(2, [2,3,4,5]).  
true ;  
false.  
  
?- member(a, [2,3,4,5,a]).  
true.
```

Figure 4.12 `member` in Prolog

- xi. Predicate `append/3`: `append(?List1, ?List2, ?List1AndList2)`
- `List1AndList2` is the concatenation of `List1` and `List2`



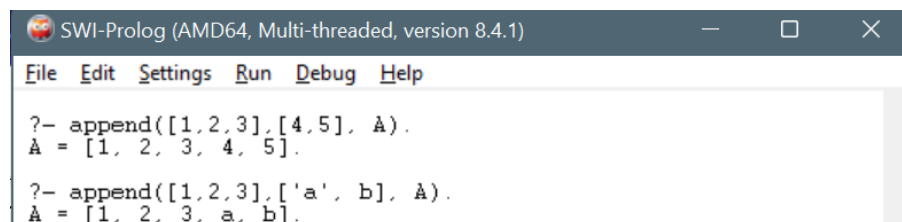
```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- append([1,2,3],[4,5], A).
A = [1, 2, 3, 4, 5].

?- append([1,2,3],['a', b], A).
A = [1, 2, 3, a, b].

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

Figure 4.13 *append in Prolog*

- xii. Predicate `append/2`: `append(+ListOfLists, ?List)`
- Concatenate a list of lists. Is true if `ListOfLists` is a list of lists, and `List` is the concatenation of these lists.

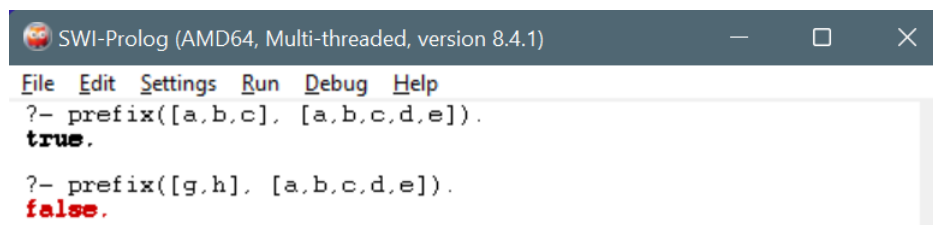


```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- append([1,2,3],[4,5], A).
A = [1, 2, 3, 4, 5].

?- append([1,2,3],['a', b], A).
A = [1, 2, 3, a, b].
```

Figure 4.14 *append in Prolog*

- xiii. Predicate `prefix/2`: `prefix(?Part, ?Whole)`.
- True if `Part` is a leading substring of `Whole`. This is the same as `append(Part, _, Whole)`.

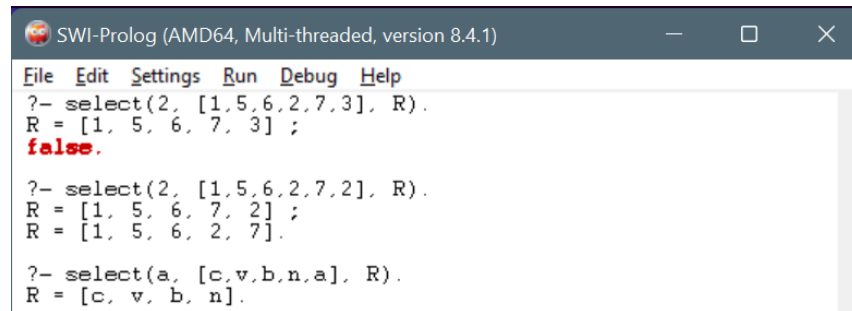


```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- prefix([a,b,c], [a,b,c,d,e]).
true.

?- prefix([g,h], [a,b,c,d,e]).
false.
```

Figure 4.15 *prefix in Prolog*

- xiv. Predicate `select/3`: `select(?Elem, ?List1, ?List2)`
- Is true when `List1`, with `Elem` removed, results in `List2`. This implementation is deterministic if the last element of `List1` has been selected.



```
File Edit Settings Run Debug Help
?- select(2, [1,5,6,2,7,3], R).
R = [1, 5, 6, 7, 3] ;
false.

?- select(2, [1,5,6,2,7,2], R).
R = [1, 5, 6, 7, 2] ;
R = [1, 5, 6, 2, 7].

?- select(a, [c,v,b,n,a], R).
R = [c, v, b, n].
```

Figure 4.16 *select in Prolog*

- xv. Predicate `reverse/2`: `reverse(?List1, ?List2)`
- Is true when the elements of `List2` are in reverse order compared to `List1`.



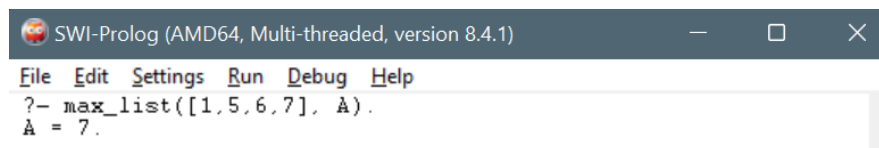
```
File Edit Settings Run Debug Help
?- reverse([1,2,3], [3,2,1]).
true.

?- reverse([1,2,3], [1,2,3]).
false.

?- reverse([1,2,3], A).
A = [3, 2, 1].
```

Figure 4.17 *reverse in Prolog*

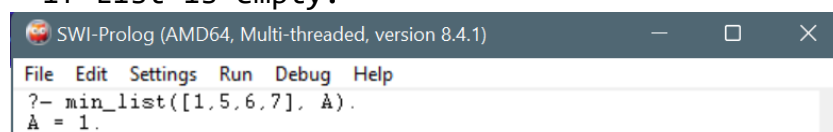
- xvi. Predicate `max_list/2`: `max_list(+List:list(number), -Max:number)`
- True if `Max` is the largest number in `List`. Fails if `List` is empty.



```
File Edit Settings Run Debug Help
?- max_list([1,5,6,7], A).
A = 7.
```

Figure 4.18 *max_list in Prolog*

- xvii. Predicate `min_list/2`: `min_list(+List:list(number), -Min:number)`
- True if `Min` is the smallest number in `List`. Fails if `List` is empty.



```
File Edit Settings Run Debug Help
?- min_list([1,5,6,7], A).
A = 1.
```

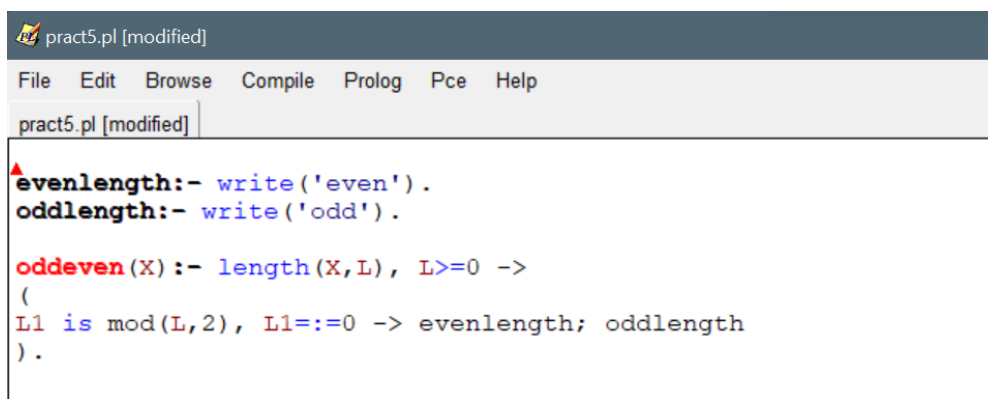
Figure 4.19 *min_list in Prolog*

AIM OF THE EXPERIMENT:

- WAP in prolog to implement two predicates `evenlength(List)` and `oddlength(List)` so that they are true if their argument is a list of even or odd respectively.
- WAP in prolog to perform append, delete and replace using list.
- WAP in prolog to count up from a number to 10.

SOURCE CODE:

- WAP in prolog to implement two predicates `evenlength(List)` and `oddlength(List)` so that they are true if their argument is a list of even or odd respectively.

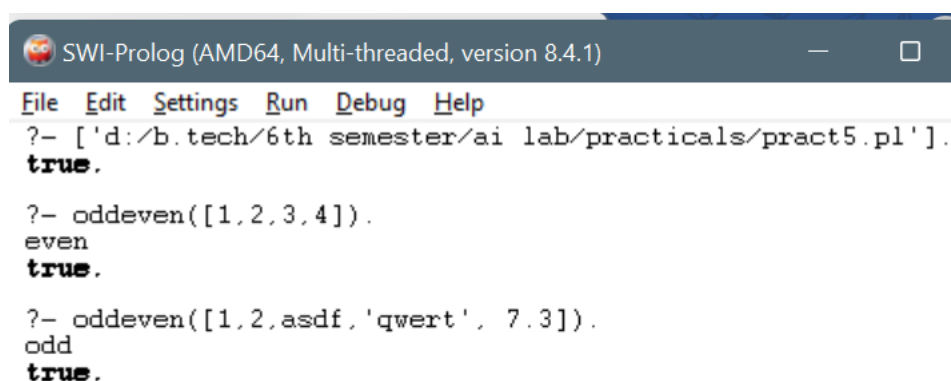


```
pract5.pl [modified]
File Edit Browse Compile Prolog Pce Help
pract5.pl [modified]
^
evenlength:- write('even').
oddlength:- write('odd').

oddeven(X):- length(X,L), L>=0 ->
(
L1 is mod(L,2), L1:=0 -> evenlength; oddlength
).
```

Fig 5.1 Program to check if the number of elements in list are even or odd

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- ['d:/b.tech/6th semester/ai lab/practicals/pract5.pl'].
true.

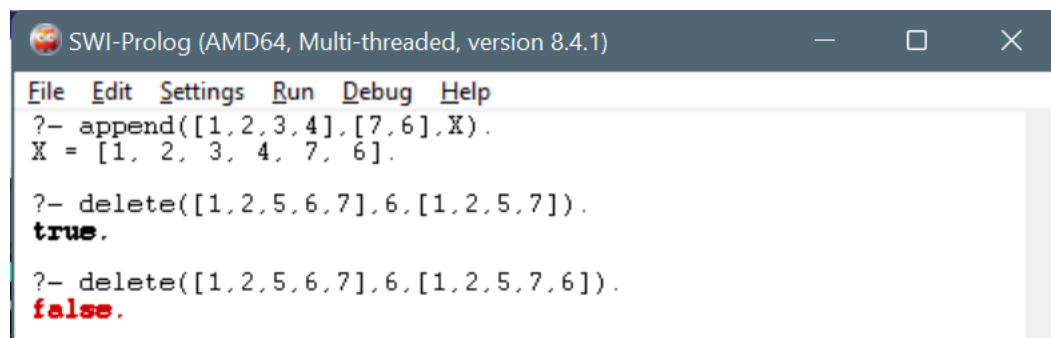
?- oddeven([1,2,3,4]).
even
true.

?- oddeven([1,2,asdf,'qwert', 7.3]).
odd
true.
```

Fig 5.2 Output

SOURCE CODE:

- b. WAP in prolog to perform append, delete and replace using list.



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- append([1,2,3,4],[7,6],X).
X = [1, 2, 3, 4, 7, 6].

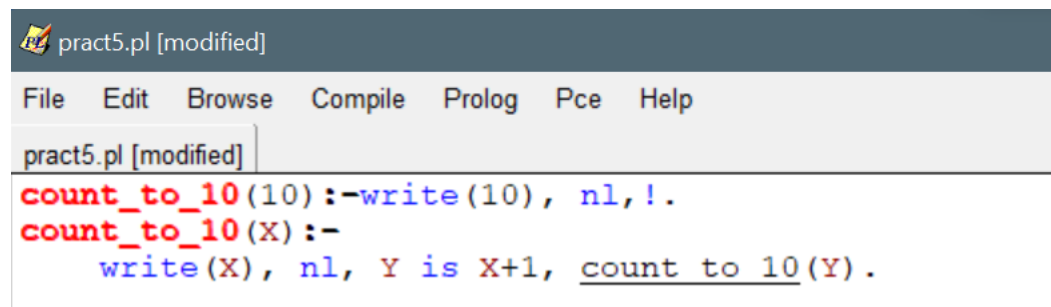
?- delete([1,2,5,6,7],6,[1,2,5,7]).
true.

?- delete([1,2,5,6,7],6,[1,2,5,7,6]).
false.
```

Fig 5.3 Append, Delete, Replace

SOURCE CODE:

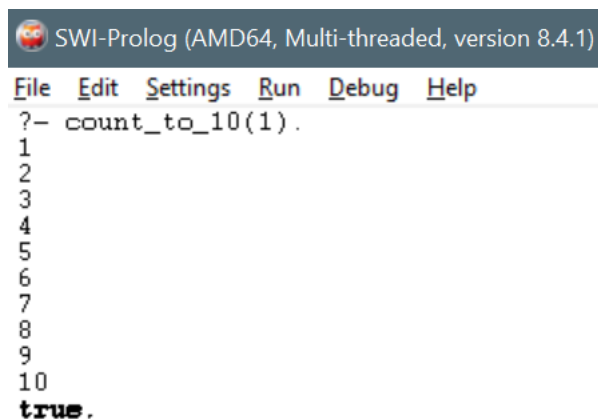
- c. WAP in prolog to count up from a number to 10.



```
pract5.pl [modified]
File Edit Browse Compile Prolog Pce Help
pract5.pl [modified]
count_to_10(10):-write(10), nl,!.
count_to_10(X):-
    write(X), nl, Y is X+1, count_to_10(Y).
```

Fig 5.5 Program to count from a number up to 10

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- count_to_10(1).
1
2
3
4
5
6
7
8
9
10
true.
```

Fig 5.6 Output

AIM OF THE EXPERIMENT:

Write a program in prolog to implement best first search algorithm.

SOURCE CODE:

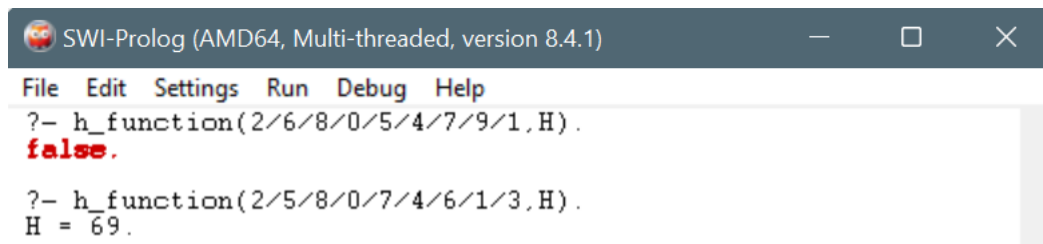
```
pract6.pl
File Edit Browse Compile Prolog Pce Help
pract7.pl pract8.pl pract6.pl
left( A/0/C/D/E/F/H/I/J , 0/A/C/D/E/F/H/I/J ).
left( A/B/C/D/0/F/H/I/J , A/B/C/0/D/F/H/I/J ).
left( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/0/H/J ).
left( A/B/0/D/E/F/H/I/J , A/0/B/D/E/F/H/I/J ).
left( A/B/C/D/E/0/H/I/J , A/B/C/D/0/E/H/I/J ).
left( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/F/H/0/I ).
;
up( A/B/C/0/E/F/H/I/J , 0/B/C/A/E/F/H/I/J ).
up( A/B/C/D/0/F/H/I/J , A/0/C/D/B/F/H/I/J ).
up( A/B/C/D/E/0/H/I/J , A/B/0/D/E/C/H/I/J ).
up( A/B/C/D/E/F/0/I/J , A/B/C/0/E/F/D/I/J ).
up( A/B/C/D/E/F/H/0/J , A/B/C/D/0/F/H/E/J ).
up( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/0/H/I/F ).
;
right( A/0/C/D/E/F/H/I/J , A/C/0/D/E/F/H/I/J ).
right( A/B/C/D/0/F/H/I/J , A/B/C/D/F/0/H/I/J ).
right( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/H/J/0 ).
right( 0/B/C/D/E/F/H/I/J , B/0/C/D/E/F/H/I/J ).
right( A/B/C/0/E/F/H/I/J , A/B/C/E/0/F/H/I/J ).
right( A/B/C/D/E/F/0/I/J , A/B/C/D/E/F/I/0/J ).
;
down( A/B/C/0/E/F/H/I/J , A/B/C/H/E/F/0/I/J ).
down( A/B/C/D/0/F/H/I/J , A/B/C/D/I/F/H/0/J ).
down( A/B/C/D/E/0/H/I/J , A/B/C/D/E/J/H/I/0 ).
down( 0/B/C/D/E/F/H/I/J , D/B/C/0/E/F/H/I/J ).
down( A/0/C/D/E/F/H/I/J , A/E/C/D/0/F/H/I/J ).
down( A/B/0/D/E/F/H/I/J , A/B/F/D/E/0/H/I/J ).
;
h_function(Puzz,H) :- p_fcn(Puzz,P),s_fcn(Puzz,S),H is P + 3*S.
move(P,C,left) :- left(P,C).
```

Fig 6.1 Source Code

```
move(P,C,up) :- up(P,C).
move(P,C,right) :- right(P,C).
move(P,C,down) :- down(P,C).
p_fcn(A/B/C/D/E/F/G/H/I, P) :- a(A,Pa), b(B,Pb), c(C,Pc),d(D,Pd), e(E,Pe), f(F,Pf),g(G,Pg), h(H,Ph), i(I,Pi),P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pi.
a(0,0). a(1,0). a(2,1). a(3,2). a(4,3). a(5,4). a(6,3). a(7,2). a(8,1).
b(0,0). b(1,0). b(2,0). b(3,1). b(4,2). b(5,3). b(6,2). b(7,3). b(8,2).
c(0,0). c(1,2). c(2,1). c(3,0). c(4,1). c(5,2). c(6,3). c(7,4). c(8,3).
d(0,0). d(1,1). d(2,2). d(3,3). d(4,2). d(5,3). d(6,2). d(7,2). d(8,0).
e(0,0). e(1,2). e(2,1). e(3,2). e(4,1). e(5,2). e(6,1). e(7,2). e(8,1).
f(0,0). f(1,3). f(2,2). f(3,1). f(4,0). f(5,1). f(6,2). f(7,3). f(8,2).
g(0,0). g(1,2). g(2,3). g(3,4). g(4,3). g(5,2). g(6,2). g(7,0). g(8,1).
h(0,0). h(1,3). h(2,3). h(3,3). h(4,2). h(5,1). h(6,0). h(7,1). h(8,2).
i(0,0). i(1,4). i(2,3). i(3,2). i(4,1). i(5,0). i(6,1). i(7,2). i(8,3).
%% the out-of-cycle function
s_fcn(A/B/C/D/E/F/G/H/I, S) :- s_aux(A,B,S1), s_aux(B,C,S2), s_aux(C,F,S3),s_aux(F,I,S4), s_aux(I,H,S5), s_aux(H,G,S6),s_aux(G,D,S7), s_aux(D,A,S8), s_aux(E,S9),S is S1+S2+S3+S4+S5+S6+S7+S8+S9.
s_aux(0,0) :- !.
s_aux(_,1).
s_aux(X,Y,0) :- Y is X+1, !.
s_aux(8,1,0) :- !.
s_aux(_,_,2).
%! %%%%%%%%% h_function(2/5/8/0/7/4/6/1/3,H).
```

Fig 6.2 Source Code (continued)

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- h_function(2/6/8/0/5/4/7/9/1,H).
false.

?- h_function(2/5/8/0/7/4/6/1/3,H).
H = 69.
```

Fig 6.3 Output

AIM OF THE EXPERIMENT:

- A. WRITE A PROGRAM IN PROLOG TO CREATE THE STATE SPACE FOR THE FOLLOWING AND CHECK THE CONNECTIONS THAT EXIST BETWEEN ANY TWO NODES**
- B. WRITE A PROGRAM IN PROLOG TO FIND THE PATH BETWEEN ANY TWO STATES OF THE FOLLOWING STATE SPACE**
- C. WRITE A PROGRAM IN PROLOG TO FIND THE PATH BETWEEN ANY TWO STATES OF THE FOLLOWING STATE SPACE.**
- D. WRITE A PROGRAM IN PROLOG TO ACCEPT THE INPUT FROM THE USER AND PERFORM STRING MATCHING.**
- E. CREATE A FAMILY TREE AND DERIVE THE RELATIONSHIPS USING PROLOG.**

SOURCE CODE

- a. Write a program in prolog to create the state space for the following and check the connections that exist between any two nodes.

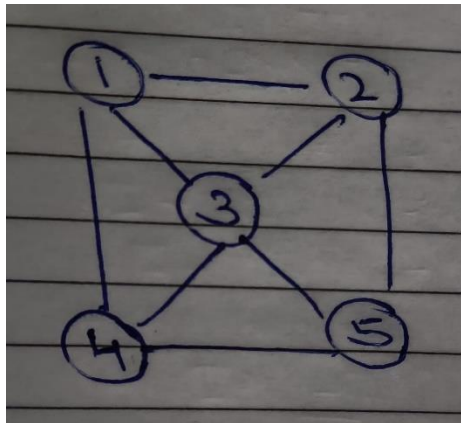


Fig 7.1 State space

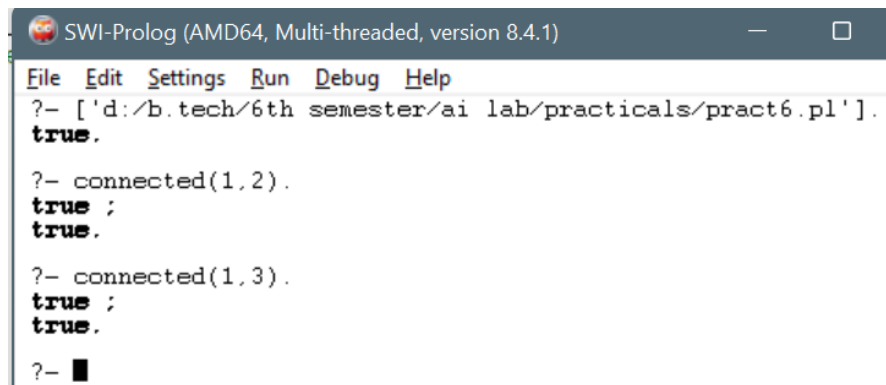
```

pract6.pl
File Edit Browse Compile Prolog Pce Help
pract6.pl pract5.pl
edge(1,2).
edge(2,1).
edge(1,3).
edge(3,1).
edge(1,4).
edge(4,1).
edge(2,3).
edge(3,2).
edge(2,5).
edge(5,2).
edge(3,4).
edge(4,3).
edge(3,5).
edge(5,3).
edge(4,5).
edge(5,4).
connected(X,Y):- edge(X,Y) ; edge(Y,X).

```

Fig 7.2 Source code

OUTPUT:

A screenshot of a SWI-Prolog terminal window. The title bar reads "SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)". The menu bar includes "File", "Edit", "Settings", "Run", "Debug", and "Help". The terminal shows the following text:

```
?- ['d:/b.tech/6th semester/ai lab/practicals/pract6.pl'].
true.

?- connected(1,2).
true ;
true.


?- connected(1,3).
true ;
true.

?-
```

Fig 7.3 Output

SOURCE CODE:

b) Write a program in prolog to find the path between any two states of the following state space (state space attached).

A screenshot of a Prolog source code editor window. The title bar reads "pract6.pl [modified]". The menu bar includes "File", "Edit", "Browse", "Compile", "Prolog", "Pce", and "Help". The editor shows the following code:

```
edge(1,2).
edge(2,1).
edge(1,3).
edge(3,1).
edge(1,4).
edge(4,1).
edge(2,3).
edge(3,2).
edge(2,5).
edge(5,2).
edge(3,4).
edge(4,3).
edge(3,5).
edge(5,3).
edge(4,5).
edge(5,4).
connected(X,Y):- edge(X,Y) ; edge(Y, X).
path(A, B, Path):- travel(A, B, [A], Q), reverse(Q, Path).
travel(A, B, P, [B|P]):- connected(A, B).
travel(A, B, Visited, Path):- connected(A, C), C=\=B, \+member(C, Visited), travel(C, B, [C|Visited], Path).
```

Fig 7.4 Source Code

OUTPUT:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- ['d:/b.tech/6th semester/ai lab/practicals/pract6.pl'].
true.

?- path(1,5, Path).
Path = [1, 2, 5] ;
Path = [1, 2, 5] ;
Path = [1, 2, 3, 5] ;
Path = [1, 2, 3, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 2, 3, 5] ;
Path = [1, 2, 3, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 3, 5] ;
Path = [1, 3, 5] ;
Path = [1, 3, 2, 5] ;
Path = [1, 3, 2, 5] ;
Path = [1, 3, 4, 5] ;
Path = [1, 3, 4, 5] ;
Path = [1, 3, 2, 5] ;
Path = [1, 3, 2, 5] ;
Path = [1, 3, 4, 5] ;
Path = [1, 3, 4, 5] ;
Path = [1, 4, 5] ;
```

Fig 7.5 Output

SOURCE CODE:

c) Write a program in prolog to find the path between any two states of the following state space

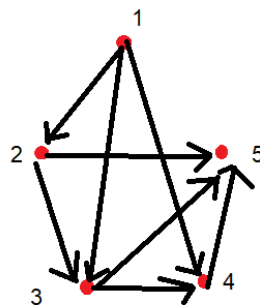


Fig 7.6 State Space

```

pract7.pl
File Edit Browse Compile Prolog Pce Help
pract7.pl
edge(1,2).
edge(1,4).
edge(1,3).
edge(2,3).
edge(2,5).
edge(3,4).
edge(3,5).
edge(4,5).
connected(X,Y):- edge(X,Y) ; edge(Y, X).
path(A, B, Path):- travel(A, B, [A], Q), reverse(Q, Path).
travel(A, B, P, [B|P]):- connected(A, B).
travel(A, B, Visited, Path):- connected(A, C), C\=B, \+member(C, Visited), travel
(C, B, [C|Visited], Path).

```

Fig 7.7 Source Code

OUTPUT:

```

SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- ['D:/B.Tech/6th semester/AI lab/practicals/pract7.pl'].
true.

?- path(1,5, Path).
Path = [1, 2, 5] ;
Path = [1, 2, 3, 5] ;
Path = [1, 2, 3, 4, 5] ;
Path = [1, 4, 5] ;
Path = [1, 4, 3, 5] ;
Path = [1, 4, 3, 2, 5] ;
Path = [1, 3, 5] ;
Path = [1, 3, 4, 5] ;
Path = [1, 3, 2, 5] ;
false.

?- path(2,4, Path).
Path = [2, 3, 4] ;
Path = [2, 3, 5, 4] ;
Path = [2, 3, 1, 4] ;
Path = [2, 5, 4] ;
Path = [2, 5, 3, 4] ;
Path = [2, 5, 3, 1, 4] ;
Path = [2, 1, 4] ;
Path = [2, 1, 3, 4] ;
Path = [2, 1, 3, 5, 4] ;
false.

```

Fig 7.8 State Space

SOURCE CODE:

d) Write a program in prolog to accept the input from the user and perform string matching.

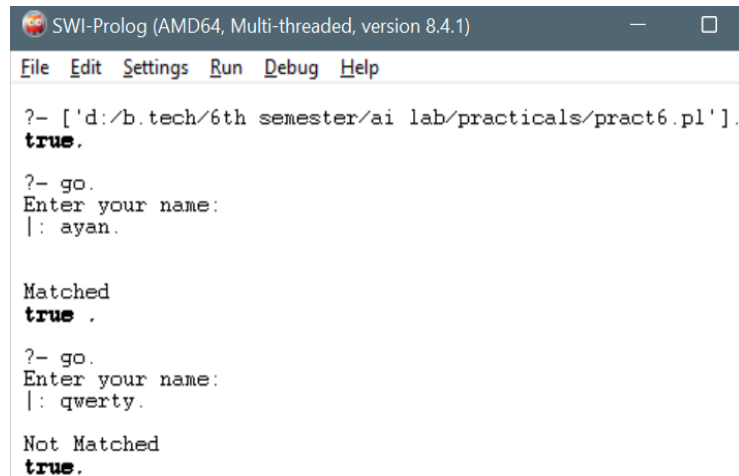
```

pract6.pl [modified]
File Edit Browse Compile Prolog Pce Help
pract6.pl [modified] pract5.pl
string1(ayan).
go:-write("Enter your name:"), nl, read(X), nl, string1(Y),
X =@= Y, nl, write("Matched") ; write("Not Matched").

```

Fig 7.9 Source Code

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help

?- ['d:/b.tech/6th semester/ai lab/practicals/pract6.pl'].
true.

?- go.
Enter your name:
|: ayan.

Matched
true.

?- go.
Enter your name:
|: qwerty.

Not Matched
true.
```

Fig 7.10 Output

SOURCE CODE:

e) Create a family tree and derive the relationships using Prolog (state space attached for reference, please create your own family tree).



```
pract7.pl
File Edit Browse Compile Prolog Pce Help

female('Mary Price').
female('Jessica Smith').
female('Angela Price').
female('Vanessa Smith').
female('Linda Price').

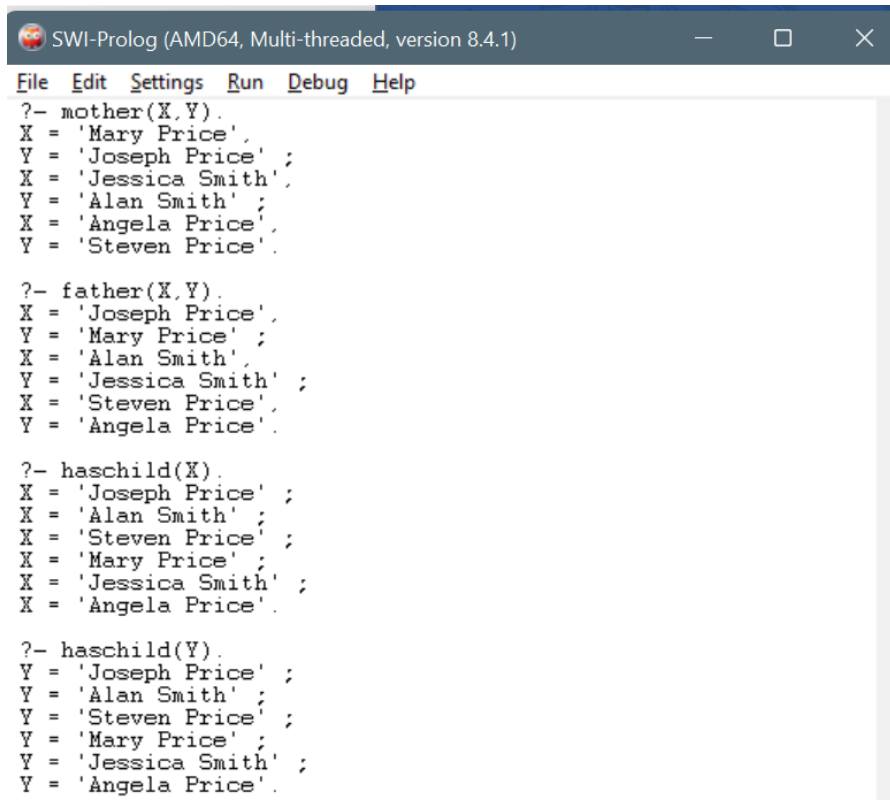
male('Joseph Price').
male('Alan Smith').
male('Steven Price').
male('Justin Smith').
male('Michael Smith').

parents('Joseph Price', 'Mary Price').
parents('Alan Smith', 'Jessica Smith').
parents('Steven Price', 'Angela Price').

mother(X,Y):- parents(Y,X),female(X).
father(X,Y):- parents(X,Y),male(X).
haschild(X):- parents(X,_).
haschild(Y):- parents(_,Y).
```


Fig 7.11 Source code

OUTPUT:



```
?- mother(X,Y).
X = 'Mary Price',
Y = 'Joseph Price' ;
X = 'Jessica Smith',
Y = 'Alan Smith' ;
X = 'Angela Price',
Y = 'Steven Price'.

?- father(X,Y).
X = 'Joseph Price',
Y = 'Mary Price' ;
X = 'Alan Smith',
Y = 'Jessica Smith' ;
X = 'Steven Price',
Y = 'Angela Price'.

?- haschild(X).
X = 'Joseph Price' ;
X = 'Alan Smith' ;
X = 'Steven Price' ;
X = 'Mary Price' ;
X = 'Jessica Smith' ;
X = 'Angela Price'.

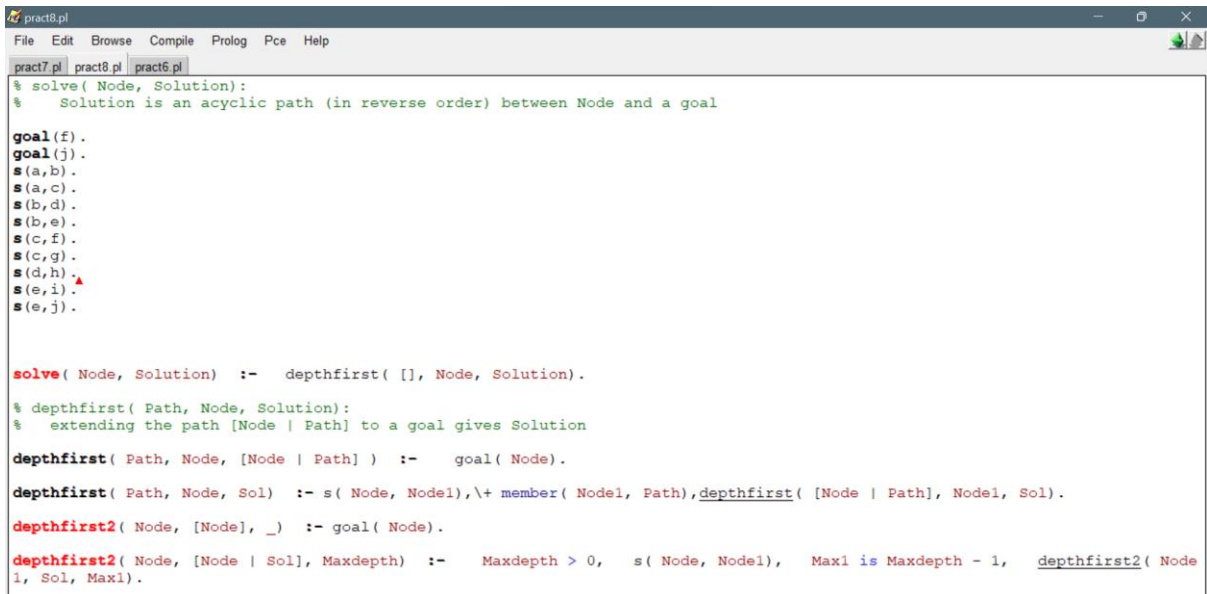
?- haschild(Y).
Y = 'Joseph Price' ;
Y = 'Alan Smith' ;
Y = 'Steven Price' ;
Y = 'Mary Price' ;
Y = 'Jessica Smith' ;
Y = 'Angela Price'.
```

Fig 7.12 Output

AIM OF THE EXPERIMENT:

Write a program in prolog to implement depth first search algorithm.

SOURCE CODE:



```
pract8.pl
File Edit Browse Compile Prolog Pce Help
pract7.pl pract8.pl pract6.pl
% solve( Node, Solution):
%   Solution is an acyclic path (in reverse order) between Node and a goal

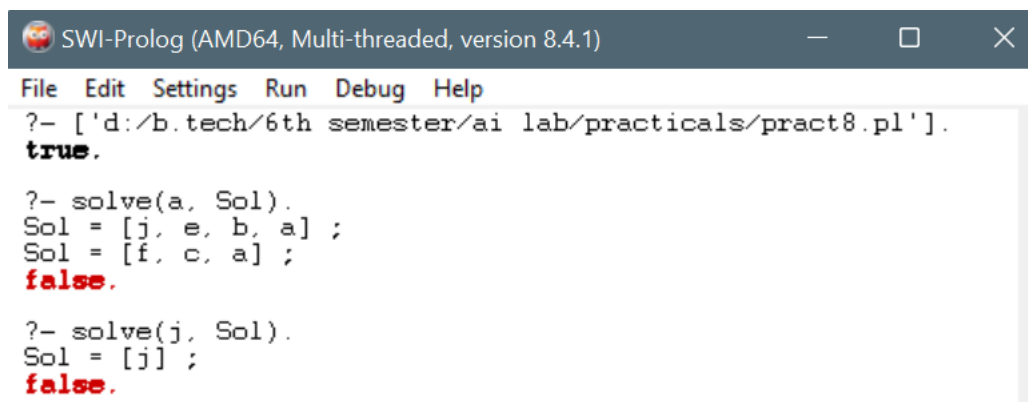
goal(f).
goal(j).
s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
s(c,g).
s(d,h).
s(e,i).
s(e,j).

solve( Node, Solution) :- depthfirst( [], Node, Solution).

% depthfirst( Path, Node, Solution):
%   extending the path [Node | Path] to a goal gives Solution
depthfirst( Path, Node, [Node | Path] ) :- goal( Node).
depthfirst( Path, Node, Sol) :- s( Node, Node1), \+ member( Node1, Path), depthfirst( [Node | Path], Node1, Sol).
depthfirst2( Node, [Node], _ ) :- goal( Node).
depthfirst2( Node, [Node | Sol], Maxdepth) :- Maxdepth > 0, s( Node, Node1), Max1 is Maxdepth - 1, depthfirst2( Node1, Sol, Max1).
```

Fig 8.1 Source Code

OUTPUT:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
File Edit Settings Run Debug Help
?- ['d:/b.tech/6th semester/ai lab/practicals/pract8.pl'].
true.

?- solve(a, Sol).
Sol = [j, e, b, a] ;
Sol = [f, c, a] ;
false.

?- solve(j, Sol).
Sol = [j] ;
false.
```

Fig 8.2 Output