ARTIFICIAL INTELLIGENCE PROGRAMS

Q1. Sum of 2 numbers

Source code:

```
sum(X,Y):- S is X+Y,write(S).
```

Output:

```
% f:/TANYA CSC 17/AI/
[1]  ?- sum(5,7).
12
true.

[1]  ?- sum(2,3).
5
true.

[1]  ?-
```

Q2. Max of 2 numbers

Source code:

```
max(X,Y,M):- X>Y, M is X.

max(_,Y,M):- M is Y.
```

Output:

```
% f:/TANYA CSC 17/AI/Q2.
[1]  ?- max(5,7,Y).
Y = 7.

[1]  ?- max(8,3,X).
X = 8 .

[1]  ?- max(2,3).
```

Q3. Factorial of a number

Source code:

```
fact(0,1).

fact(N,R):-N1 is N-1, fact(N1,R1), R is N*R1.
```

Output:

```
% f:/TANYA CSC 17/AI/Q3
[1]  ?- fact(5,R).
R = 120 .

[1]  ?- fact(4,X).
X = 24 .

[1]  ?-
```

Q4. Nth term of fibonacci series

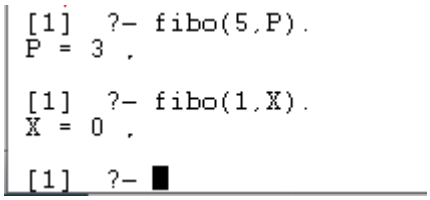
Source code:

fibo(1,0).

fibo(2,1).

fibo(P,N):- P1 is P-1,fibo(P1,R1),P2 is P-2,fibo(P2,R2),N is R1+R2.

Output:



Q5. GCD of two numbers

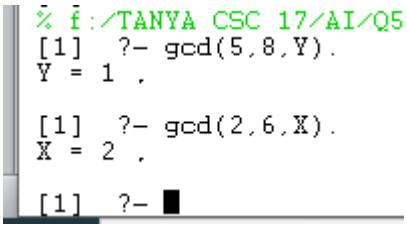
Source code:

gcd(X,X,X).

gcd(X,Y,D):- X<Y, gcd(Y,X,D).

gcd(X,Y,D):- X>Y,Y1 is X-Y,gcd(Y,Y1,D).

Output:



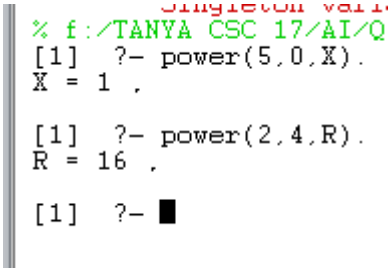
Q6. Power of a number

Source code:

power(Num,0,Ans):-Ans is 1.

power(Num,Pow,Ans):-P is Pow-1,power(Num,P,Ans1),Ans is Ans1*Num.

Output:



Q7. Multiplication of two numbers

Source code:

multi(N1,1,N1).

multi(N1,N2,Ans):- Temp is N2-1, multi(N1,Temp,Ans1),Ans is Ans1+N1.

Output:

```
% f:/TANYA CSC 17/AI/Q7.pl
[1]  ?- multi(5,6,R).
R = 30 .

[1]  ?- multi(2,3,X).
X = 6 .

[1]  ?-
```

Q8. Tower of Hanoi.

Source code:

mov(0,_,_,_):-!.

mov(N,L,C,R):- N1 is N-1,mov(N1,L,R,C),write("Move disk "),write(N),write(" from "),write(L),write(" to "),write(C),nl,mov(N1,C,L,R).

toh(N):-mov(N,left,center,right).

%L is treated as Source,R as destination and C as intermediate peg.

Output:

```
?- toh(3).
Move disk 1 from left to center
Move disk 2 from left to right
Move disk 1 from right to left
Move disk 3 from left to center
Move disk 1 from center to right
Move disk 2 from center to left
Move disk 1 from left to center
true.
```

Q9. Cyclic directed graph.

Source code:

edge(p,q).

%edge(q,r).

edge(q,s).

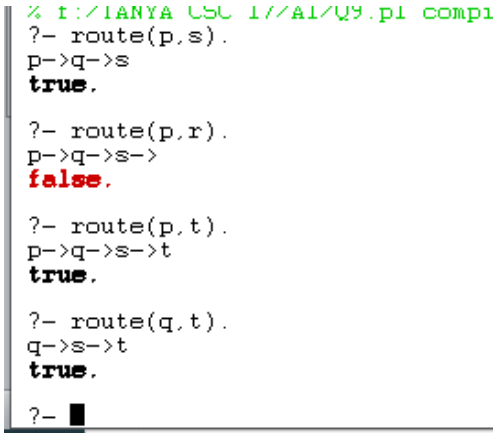
edge(s,t).

route(X,X):-write(X),!.

%route(X,Y):-edge(X,Y),write(X),write("->"),write(Y),!.

route(X,Y):-edge(X,Z),write(X),write("->"),route(Z,Y).

Output:



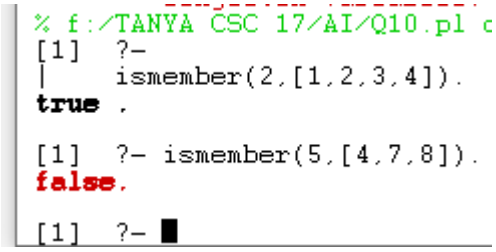
Q10. X is a member of list or not.

Source code:

ismember(X,[X|T]).

ismember(X,[H|T]):-ismember(X,T).

Output:



Q11. Concatenate 2 lists and store in 3rd list.

Source code:

conc([],L,L).

conc([H|T],L2,[H|L3]):-conc(T,L2,L3).

Output:

```
[1]  ?- conc(X,Y,[c,d]).
X = [],
Y = [c, d] ;
X = [c],
Y = [d] ;
X = [c, d],
Y = [] ;
false.

[1]  ?- conc([a,b],[c,d],L).
L = [a, b, c, d].

[1]  ?- conc([],[c,d],L).
L = [c, d].

[1]  ?- conc([a,b],[],L).
L = [a, b].

[1]  ?- conc(X,Y,[a,b,c,d]).
X = [],
Y = [a, b, c, d] ;
X = [a],
Y = [b, c, d] ;
X = [a, b],
Y = [c, d] ;
X = [a, b, c],
Y = [d] ;
X = [a, b, c, d],
Y = [] ;
false.

[1]  ?-
```

```
[trace]  ?- conc(X,Y,[c,d]).
   Call: (8) conc(_4198, _4200, [c, d]) ? creep
   Exit: (8) conc([], [c, d], [c, d]) ? creep
X = [],
Y = [c, d] ;
   Redo: (8) conc(_4198, _4200, [c, d]) ? creep
   Call: (9) conc(_4456, _4200, [d]) ? creep
   Exit: (9) conc([], [d], [d]) ? creep
   Exit: (8) conc([c], [d], [c, d]) ? creep
X = [c],
Y = [d] ;
   Redo: (9) conc(_4456, _4200, [d]) ? creep
   Call: (10) conc(_4462, _4200, []) ? creep
   Exit: (10) conc([], [], []) ? creep
   Exit: (9) conc([d], [], [d]) ? creep
   Exit: (8) conc([c, d], [], [c, d]) ? creep
X = [c, d],
Y = [] ;
   Redo: (10) conc(_4462, _4200, []) ? creep
   Fail: (10) conc(_4462, _4200, []) ? creep
   Fail: (9) conc(_4456, _4200, [d]) ? creep
   Fail: (8) conc(_4198, _4200, [c, d]) ? creep
false.

[trace]  ?-
```

Commands to go back from trace.

```
[trace] [1]  ?- notrace.
true.

[debug] [1]  ?- nodebug.
true.
```

Q12. Reverse a list.

Source code:

```
conc([],L,L).

conc([H|T],L2,[H|L3]):-conc(T,L2,L3).


revlist([],[]).

revlist([H|T],R):-revlist(T,Trev),conc(Trev,[H],R).
```

Output:

```
?- revlist([1,2,3,4],R).
R = [4, 3, 2, 1].

?- revlist([],R).
R = [].

?- revlist([1],R).
R = [1].

?-
```

Q13. Check whether a list is palindrome or not.

Source code:

```
conc([],L,L).

conc([H|T],L2,[H|L3]):-conc(T,L2,L3).


revlist([],[]).

revlist([H|T],R):-revlist(T,Trev),conc(Trev,[H],R).


palind(L):-revlist(L,R), L=R -> write("It is a Palindrome");write("It is not a palindrome").
```

Output:

```
% f:/TANYA CSC 17/AI/Q13.pl
?- palind([1,2,1]).
It is a Palindrome
true.

?- palind([1,2,1,2]).
It is not a palindrome
true.

?- palind([]).
It is a Palindrome
true.

?-
```

Q14. Sum of elements of given list.

Source code:

```
sumlist([],0).

sumlist([H\T],S):-sumlist(T,S1),S is H+S1.
```

Output:

```
% f:/TANYA CSC 17/AI/Q14.pl com
[1]  ?- sumlist([],R).
R = 0.

[1]  ?- sumlist([1,2,3,4],R).
R = 10.

[1]  ?-
```

Q15. Even length or odd length list.

Source Code:

```
len([],0).

len([H|T],N) :- len(T,N1), N is N1+1.

evenlength(List):- len(List,N),mod(N,2)=:=0,write("The list ids even length").

oddlength(List):-len(List,N),mod(N,2)=\=0,write("The list is odd length").

check_list(L):- evenlength(L);oddlength(L).

/* ; mean either goal can be true */
```

Output:

```
% f:/TANYA CSC 17/AI/Q15.pl compiled 0.0
?- evenlength([1,2,3,4]).
The list ids even length
true.

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

?- oddlength([1,2,3]).
The list is odd length
true.

?- oddlength([1,2,3,4]).
false.

?- check_list([1,2,3,4]).
The list ids even length
true .

?-
```

Q16. Nth element in a list.

Source Code:

```
nth_element(1,[H|T],H).

nth_element(N,[H|T],X):-N1 is N-1, nth_element(N1,T,X).
```

Output:

```
% f:/TANYA CSC 17/AI/Q16.pl compiled
?- nth_element(3,[1,2,3,4],X).
X = 3 ;

?- nth_element(5,[1,2,3,4,8],X).
X = 8 ;
false.

?- nth_element(7,[1,2,3,4,8],X).
false.

?-
```

Q17. Write a Prolog program to implement remdup(L, R) to remove duplicates from a list L to generate a list R.

Source code:

```
ismember(X,[X|T]).

ismember(X,[H|T]):-ismember(X,T).

remove_dups([],[]).

remove_dups([H|T],R):-ismember(H,T),remove_dups(T,R).

remove_dups([H|T],[H|R]):-not(ismember(H,T)),remove_dups(T,R).
```

Output:

```
% c:/Users/LIPIKA/Desktop/duplicate.pl comp
?- remove_dups([a,b,c,d,a,a,a,g,b,h,d,i,c],
R = [a, g, b, h, d, i, c] .

?- remove_dups([a,b,c,d,a,a,a,g,h,d,i,c],R)
R = [b, a, g, h, d, i, c] .
```

Q18. Find maximum element in list.

Source code:

```
max(X,Y,M) :- (X=Y, M is X;

X>Y , M is X;

M is Y

).


maxlist([H|[]], H).

maxlist([H|T],M) :- maxlist(T,M1), max(H,M1,M).
```

Output:

```
% f:/TANYA CSC 17/AI/Q18.pl compiled
?- maxlist([1,2,3,4,5],M).
M = 5 .

?- maxlist([1],M).
M = 1 .

?- maxlist([9,7,12,5],M).
M = 12 .

?- maxlist([9,9],M).
M = 9 .
```

Q19. Write a prolog program to implement insert_nth(I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.

Source code :

```
insert_nth(I, 1, L, [I|L]).

insert_nth(I, N, [H|T], [H|R]):- N1 is N-1,insert_nth(I, N1, T, R).
```

Output :

```
% c:/Users/LIPIKA/Desktop/Tanya/AI/insert.pl c
.

?- insert_nth(6,1,[1,2],R).
R = [6, 1, 2] .

?- insert_nth(6,5,[1,2,3,4,7,8],R).
R = [1, 2, 3, 4, 6, 7, 8] .

?- insert_nth(3,2,[1,2,3,4,7,8],R).
R = [1, 3, 2, 3, 4, 7, 8] .

?-
```

Q20. Write a Program in PROLOG to implement sublist(S, L) that checks whether the list S is the sublist of list L or not. (Check for sequence or the part in the same order).

Source code:

```
sublist([],L).

sublist(S,[]):- false.

sublist([H1|T1],[H1|T2]):- sublist(T1,T2).

sublist([H1|T1],[H2|T2]):- sublist([H1|T1],T2).
```

Output:

```
% c:/Users/LIPIKA/Desktop/Tanya/AI/Q20.
?- sublist([],L).
true .

?- sublist([1,2],[]).
false.

?- sublist([1,2],[1,2,4,5]).
true .

?- sublist([1,2],[5,6,1,2,4,5]).
true .

?- sublist([7,2],[5,6,1,2,4,5]).
false.

?- sublist([7,8],[5,6,1,2,4,5]).
false.

?-
```

Q21. : Write a Prolog program to implement delete_nth (N, L, R) that removes the element on Nth position from a list L to generate a list R.

Source code:

```
delete_nth(1, [H|T], T).

delete_nth(N, [H|T], [H|R]):- N1 is N-1,delete_nth(N1, T, R).
```

Output:

```
% c:/Users/LIPIKA/Desktop/Tanya/AI/Q21.p
?- delete_nth(1,[2,3,4,5],R).
R = [3, 4, 5] .

?- delete_nth(4,[2,3,7,8,9,5],R).
R = [2, 3, 7, 9, 5] .

?-
```

Q22. Write a program in PROLOG to implement delete_all (X, L, R) where X denotes the element whose all occurrences has to be deleted from list L to obtain list R.

Source code:

```
delete_all(X, [], []).

delete_all(X, [X|T], R):- delete_all(X, T, R).

delete_all(X, [H|T], [H|R]):- delete_all(X, T, R).
```

Output:

```
% c:/Users/LIPIKA/Desktop/Tanya/AI/Q22.pl
?- delete_all(5,[],R).
R = [].

?- delete_all(5,[1,2,3],R).
R = [1, 2, 3].

?- delete_all(5,[1,5,2,4,5,6,5,7,5],R).
R = [1, 2, 4, 6, 7] .

?-
```

Q23. Write a program in PROLOG to implement merge (L1, L2, L3) where L1 is first ordered list and L2 is second ordered list and L3 represents the merged list.

Source code:

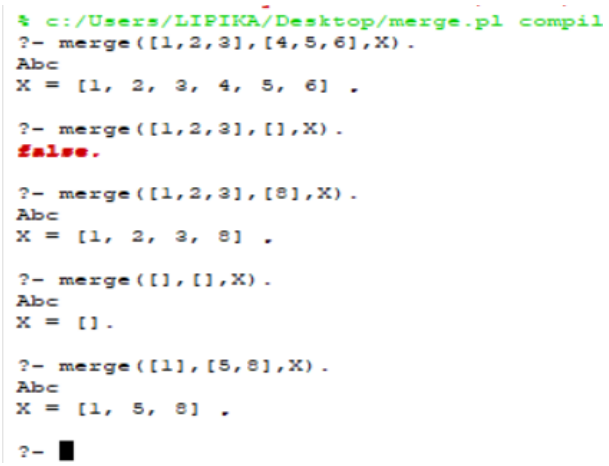
```
merge([],L,L):-!,write("Abc").

merge([H1|T1],[H2|T2],[H3|T3]):-H1>H2,H3 is H2,merge([H1|T1],T2,T3).

merge([H1|T1],[H2|T2],[H3|T3]):-H1<H2,H3 is H1,merge(T1,[H2|T2],T3).

merge([H1|T1],[H2|T2],[H3,H4|T3]):-H1=H2,H3 is H1,H4 is H2,merge(T1,T2,T3).
```

Output:



Q24. Write a PROLOG program that will take grammar rules in the following format:

NT -> (NT | T)*

Where NT is any nonterminal, T is any terminal and Kleene star (*) signifies any number of repetitions, and generate the corresponding top-down parser, that is:

sentence -> noun-phrase, verb-phrase
determiner -> [the]will generate the following:
sentence (I, O) :- noun-phrase(I,R), verb-phrase (R,O).
determiner ([the|X], X) :- !.

Source code:

```
sentence-->np,vp.

np-->det,noun.

vp-->verb.

vp-->verb,np.


det-->[a].

det-->[the].

det-->[an].
```

noun-->[boy].

noun-->[girl].

noun-->[song].

noun-->[apple].

verb-->[sing].

verb-->[sings].

verb-->[eats].

Output:

```
% c:/Users/LIPIKA/Desktop/grammar.pl con
?- sentence([the,girl,sings,a,song],[]).
true.

?- sentence([the,girl,a,song],[]).
false.
```

Q25. Write a prolog program that implements Semantic Networks.

Source code:

%Facts%

mat(mat1).

sat_on(Cat1,mat1).

cat(Cat1).

cat(tom).

color(tom,black).

owns(john,tom).

bird(bird1).

caught(tom,bird1).

is_coloured(tom,ginger).

%Rules%

animal(X):- mammal(X).

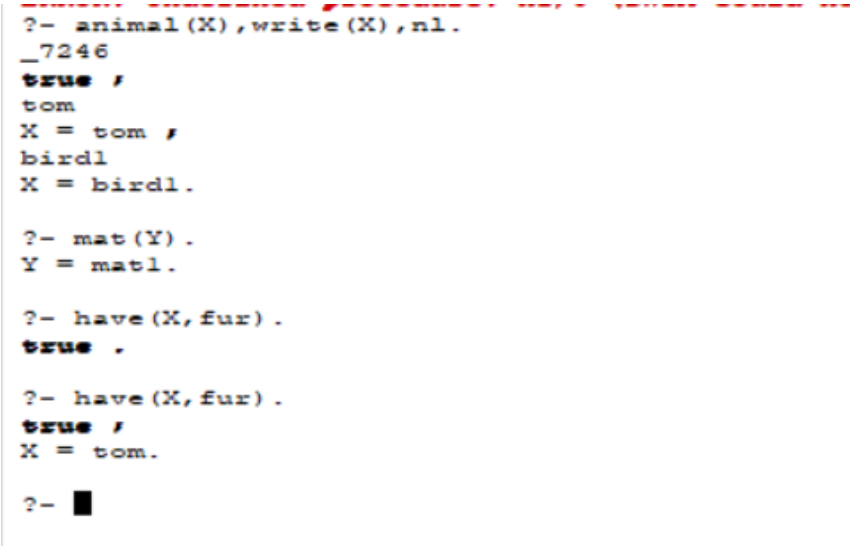
animal(X):- bird(X).

mammal(X):- cat(X).

have(X,fur):- mammal(X).

likes(X,cream):- cat(X).

Output:



Source code:

(b)

ako(mammal,animal).

ako(bird,mammal).

ako(elephant,mammal).

is_a(tweety,bird).

is_a(clyde,elephant).

is_a(neil,elephant).

can(animal,breathe).

can(animal,move).

can(bird,fly).

can(tweety,sing).

has(neil,three_legs).

has(bird,feather).

has(bird,wing).

has(elephant,four_legs).

has(elephant,one_trunk).

has(elephant,one_tail).

has(animal,skin).

has(mammal,head).

color(elephant,grey).

color(tweety,yellow).

like(neil,apple).

%Rules

check(O,P,V):- G=..[P,O,V],G,!;!.

check(O,P,V):-is_a(O,X),check(X,P,V);ako(O,X),check(X,P,V).

Output: