**Artificial Intelligence Practical Back-up**

***Vaishnavi Jha***

***2018355  
B.Sc. (Hons.) Computer Science***

***2018-2021***

**GUIDELINES QUESTIONS**

1. **Write a prolog program to calculate sum of two numbers.**

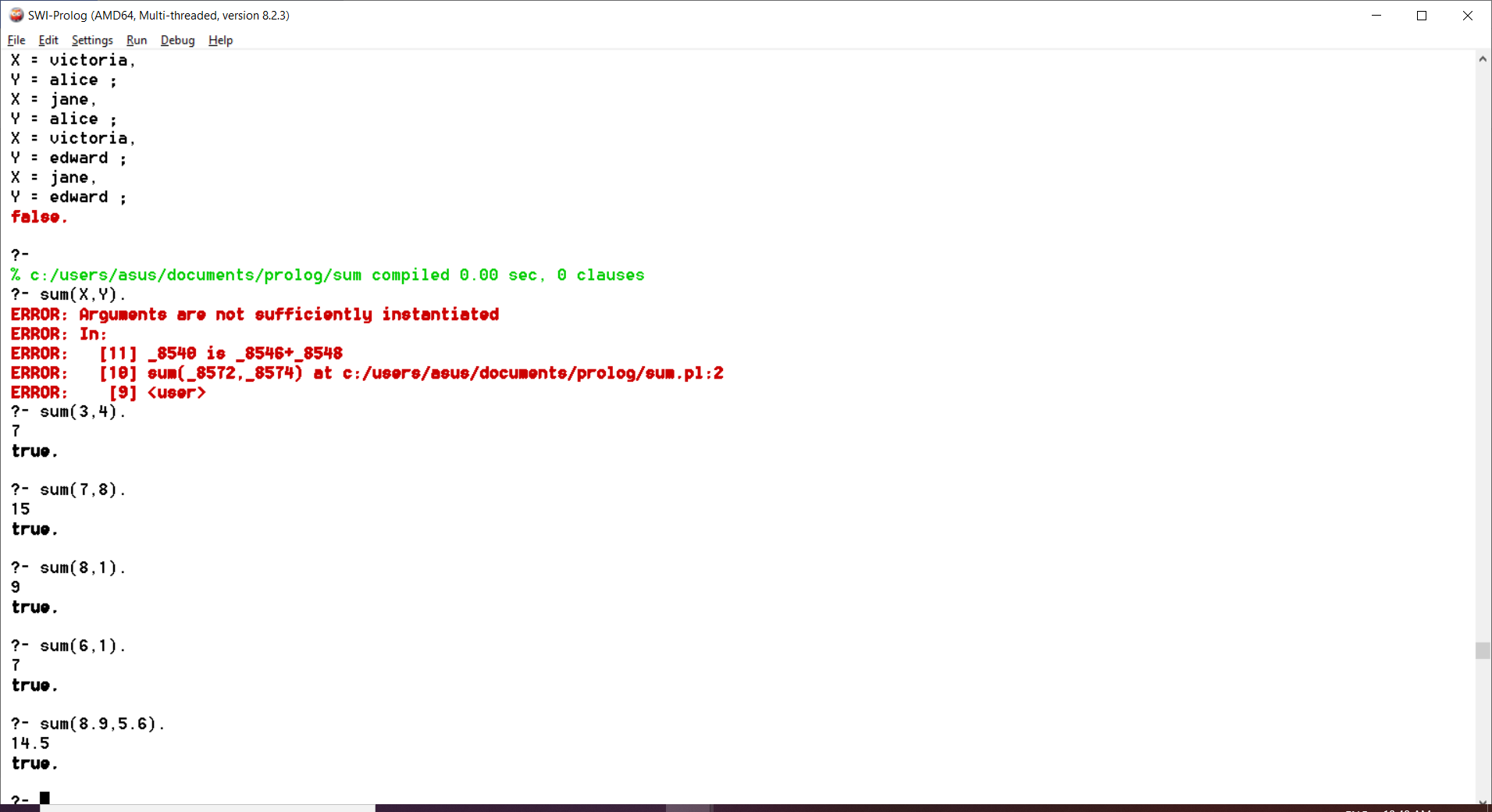
**Program:**

sum(X,Y):-

S is X+Y,

write(S).

**Output:**



1. **Write a prolog program to implement max(X,Y,M) where M is the maximum of the two numbers X and Y.**

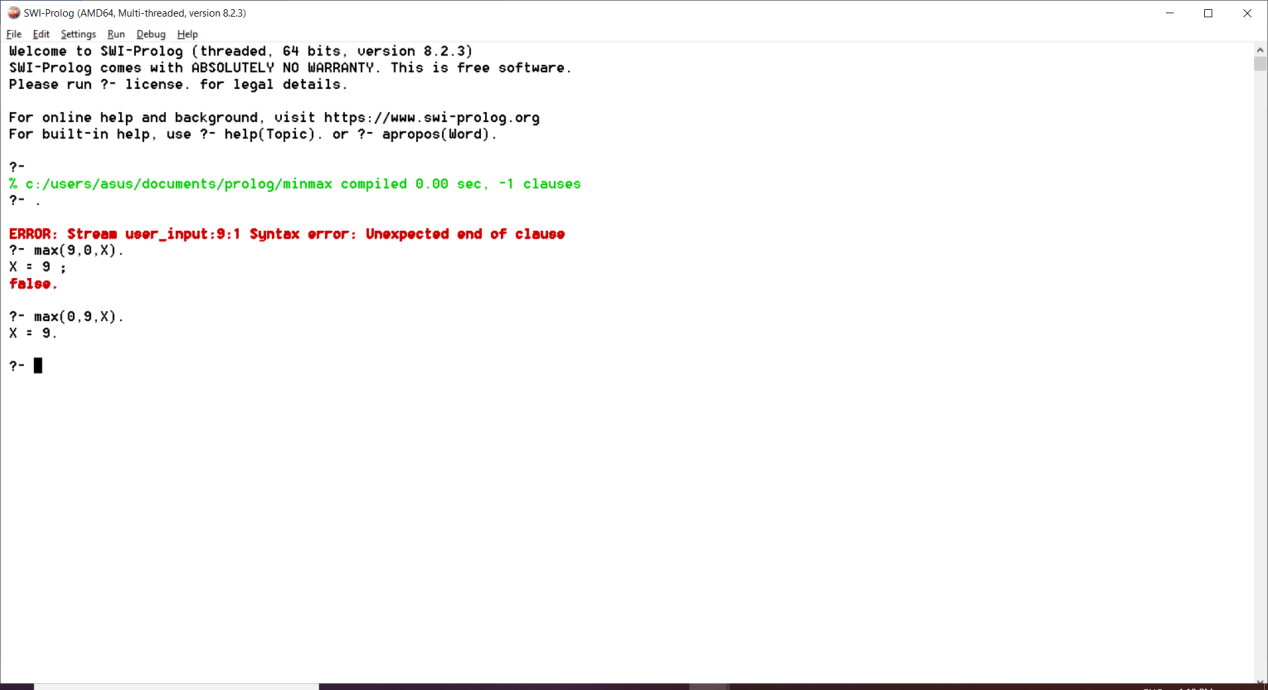
**Program:**

max(X,Y,M):-X>Y,

M is X.

max(X,Y,M):-Y>=X,

M is Y.

**Output:**

1. **Write a prolog program to implement factorial(N,F) where F is the factorial of number N.**

**Program:**

factorial(0,1).

factorial(N,F):-

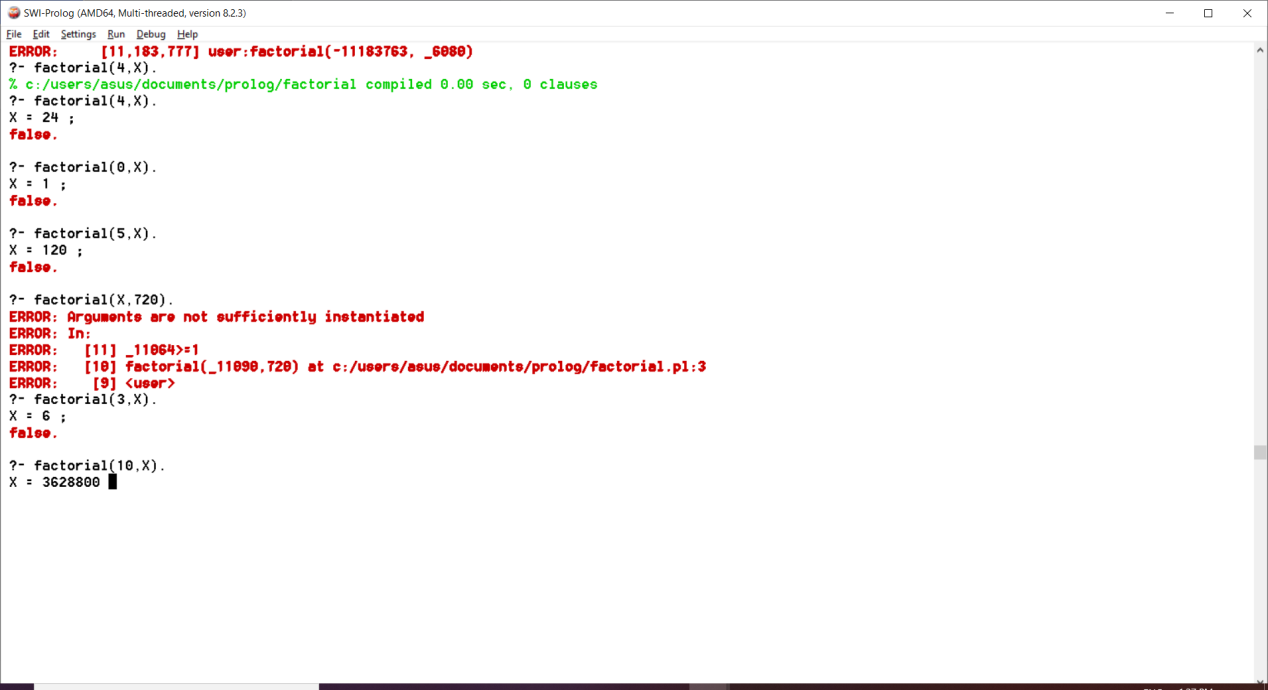
N >=1,

N1 is N-1,

factorial(N1,F1),

F is N\*F1.

**Output:**



1. **Write a prolog program to implement generate\_fib(N,T) where T represents the Nth term of the fibonacci series.**

**Program:**

fibonacci(1,0).

fibonacci(2,1).

fibonacci(N,X):-

N > 2,

N1 is N-1,

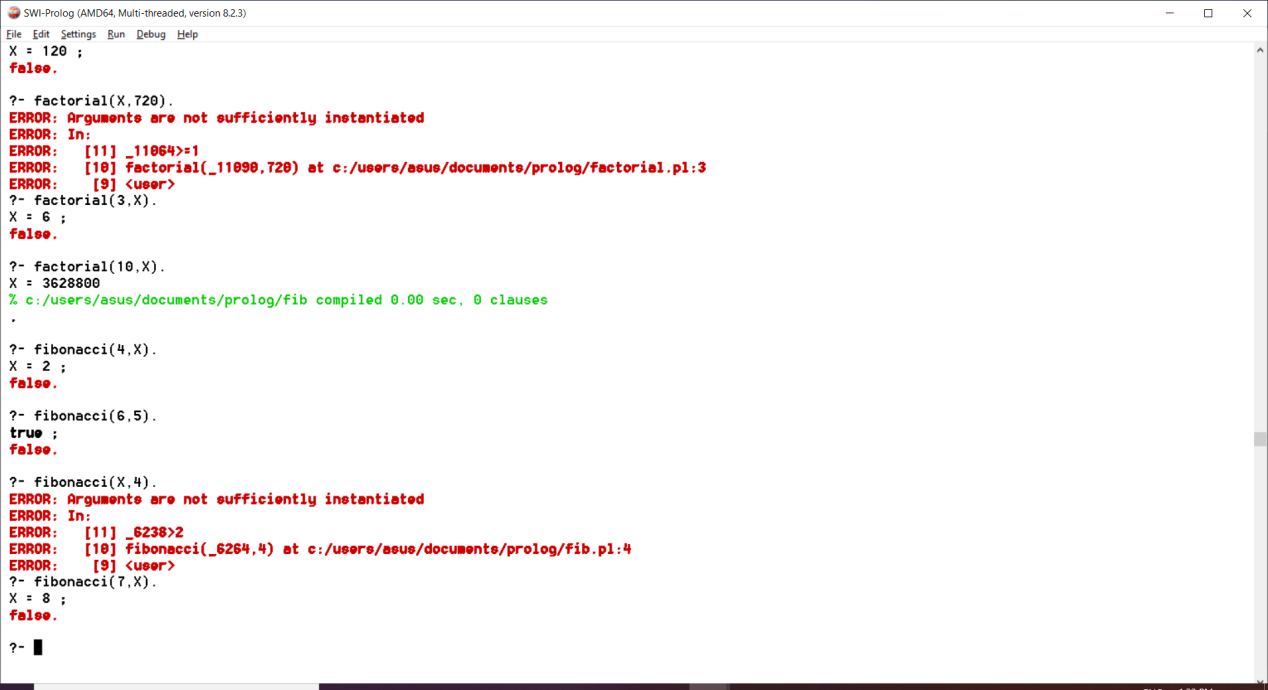
N2 is N-2,

fibonacci(N1,X1),

fibonacci(N2,X2),

X is X1+X2.

**Output:**



1. **Write a prolog program to implement GCD of two numbers.**

**Program:**

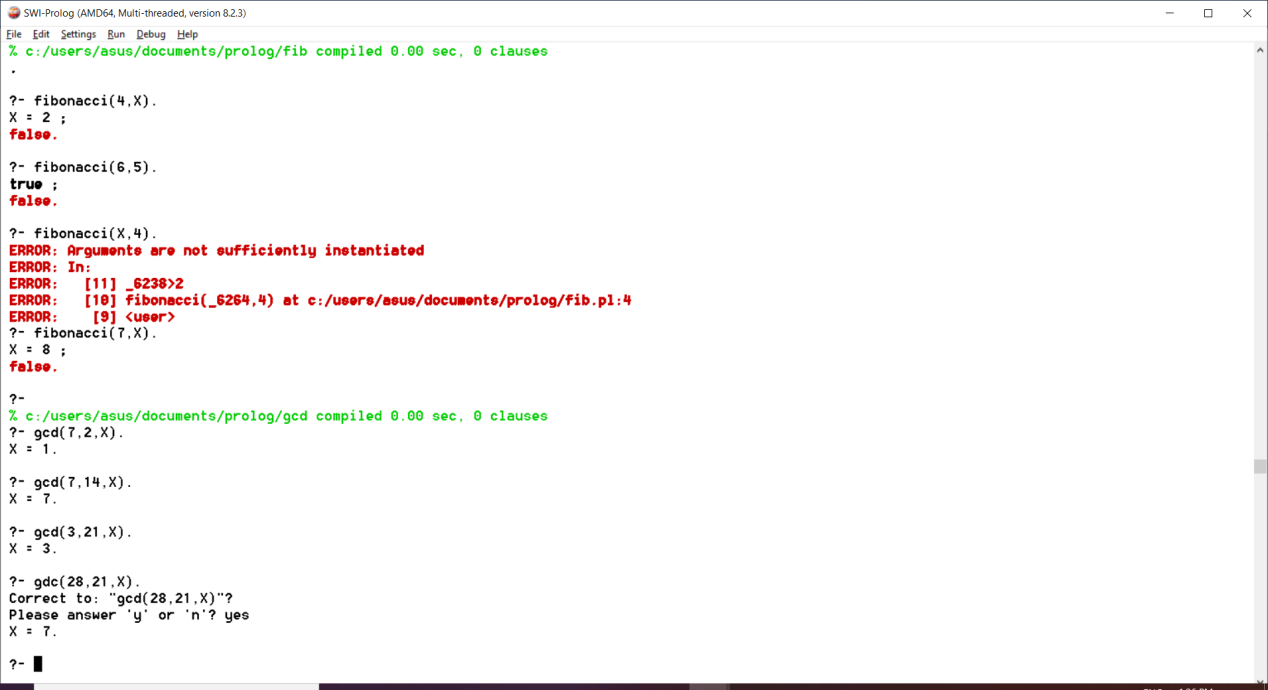
gcd(0,A,A):-!.

gcd(A,0,A):-!.

gcd(A,B,R):-B1 is mod(A,B),

gcd(B,B1,R).

**Output:**



1. **Write a prolog program to implement power(Num,Pow,Ans): where Num is raised to the Pow to get.**

**Program:**

power(\_,0):-!.

power(Num,Pow,Ans):-Ans is Num^Pow.

**Output:**

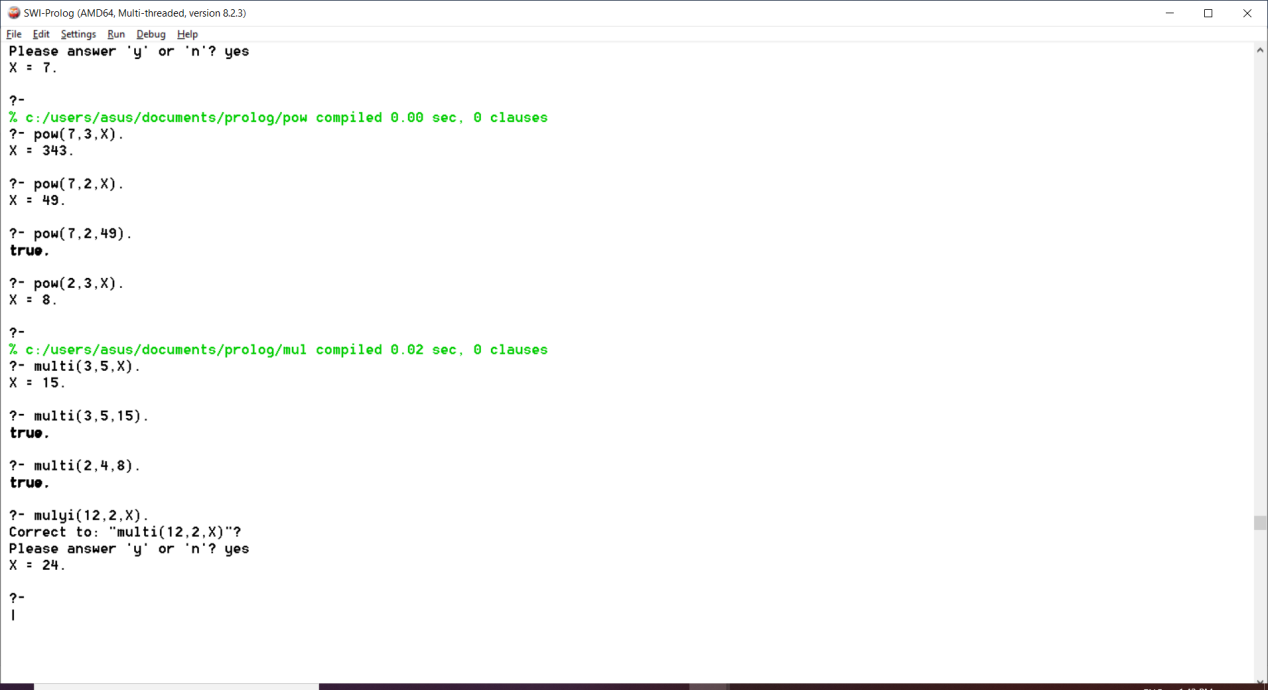


1. **Write a prolog program to implement multi(N1,N2,R): where N1 and N2 denote the numbers to be multiplied and R represents the result.**

**Program:**

multi(N1,N2,R):-R is N1\*N2.

**Output:**



1. **Write a program in PROLOG to implement towerofhanoi (N) where N represents the number of discs.**

**Program:**

towerofhanoi(N):-

move(N,"Left","Right","Center").

move(1,Left,Right,\_):-write("move one disk on the top from"),write(Left),write("to"),write(Right),nl,!.

move(N,Left,Right,Center):-N>1,

N1 is N-1,

move(N1,Left,Center,Right),

move(1,Left,Right,\_),

move(N1,Center,Right,Left),nl.

**Output:**



1. **Consider a Cyclic directed Graph – [edge(p,q),edge(q,r),edge(r,s),edge(s,t)] where edge(A,B) is a predicate indicating directed edge in graph from node A to B. WAP to check if route exists from one node to another.**

**Program:**

edge(p,q).

edge(q,r).

edge(r,s).

edge(s,t).

edge(a,b).

route(X,Y):-

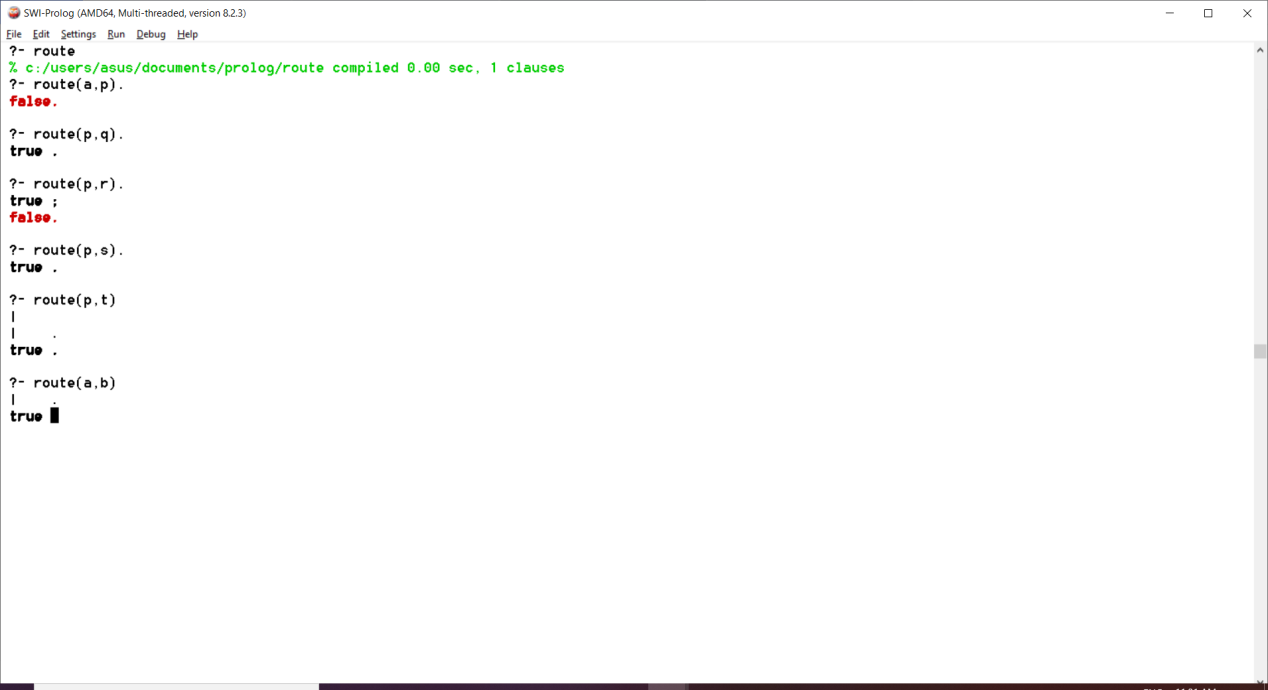
edge(X,Y).

route(X,Y):-

edge(X,Z),

route(Z,Y).

**Output:**



1. **Write a prolog program to implement memb(X,L): to check whether X is a member of L or not.**

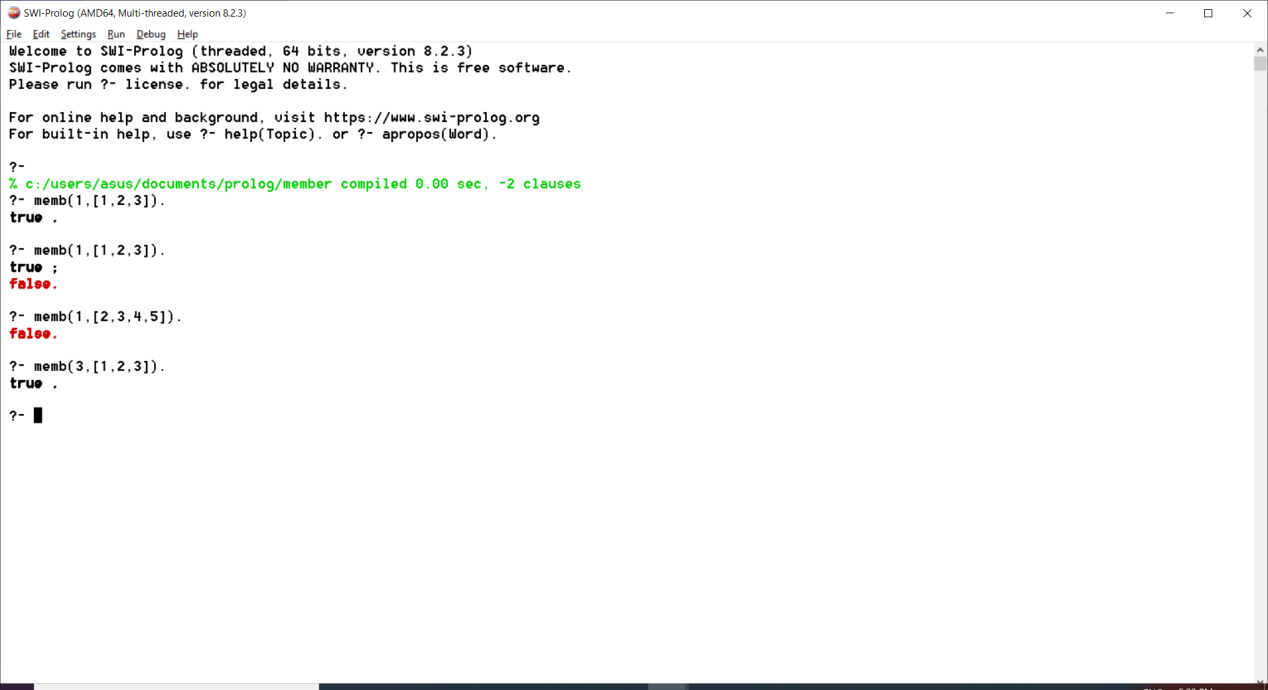
**Program:**

memb(X,[X|\_]).

memb(X,[\_|Z]):-

memb(X,Z).

**Output:**



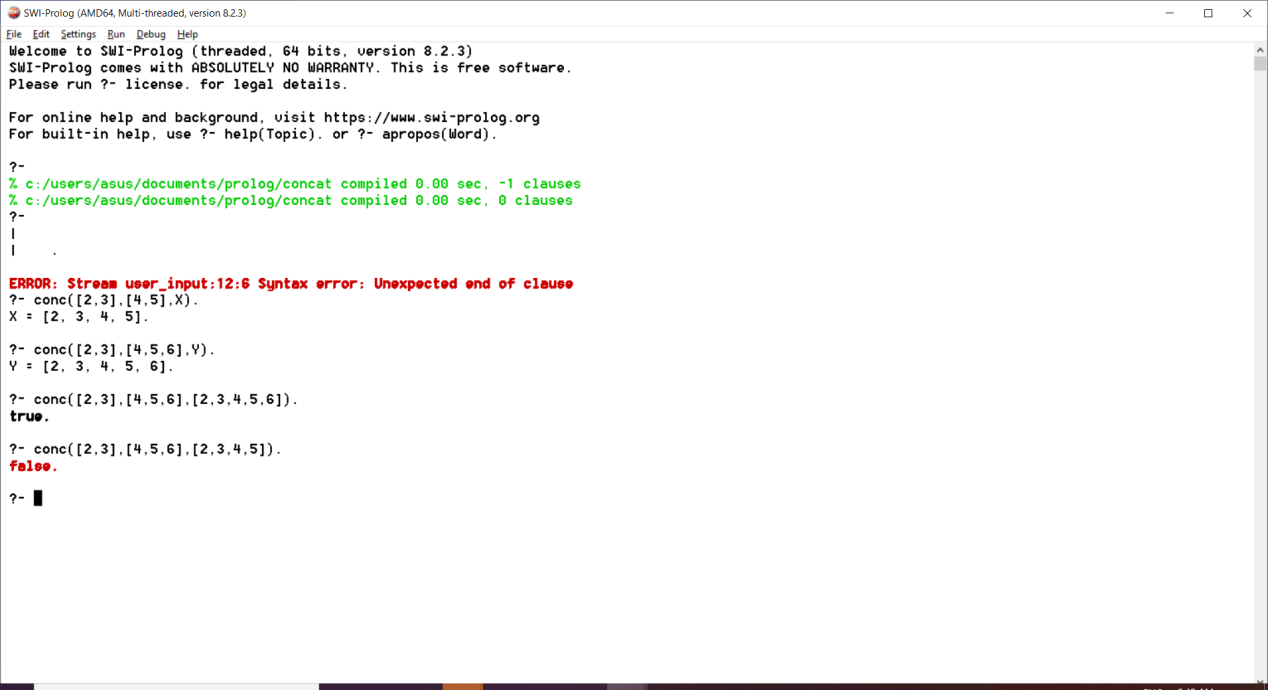
1. **Write a Prolog program to implement conc (L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.**

## **Program:**

conc([],L,L).

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

**Output:**



1. **Write a prolog program to implement reverse (L,R) where List L is original and list R is reversed list.**

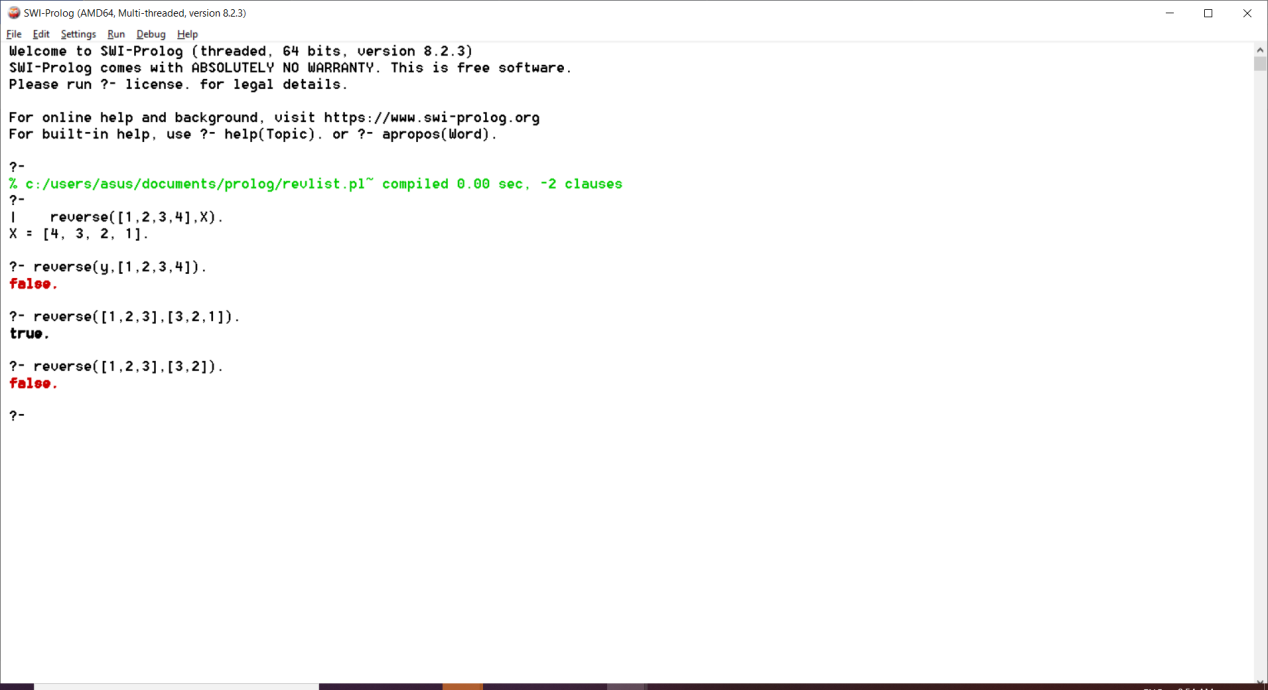
**Program:**

ar([H|T],A,R):-ar(T,[H|A],R).

ar([],A,A).

reverse(L,R):-ar(L,[],R).

**Output:**



1. **Write a prolog program Implement palindrome(L) to check whether list L is a palindrome or not.**

## **Program:**

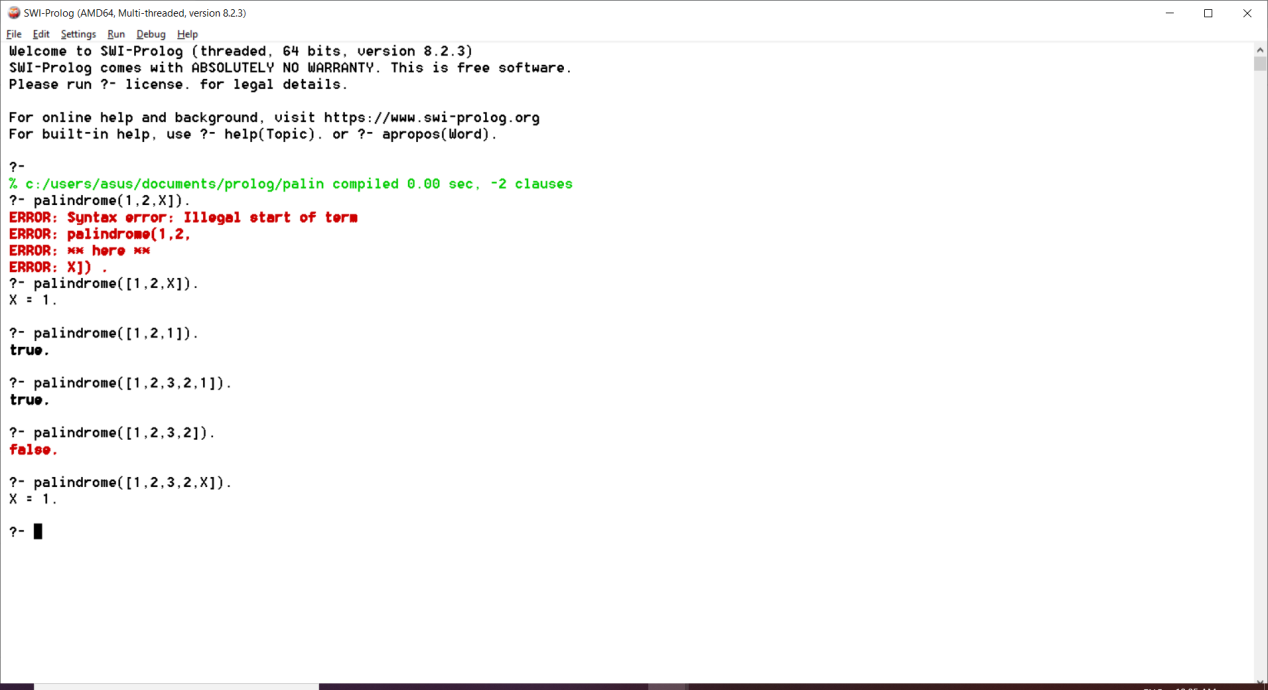
ar([H|T],A,R):-ar(T,[H|A],R).

ar([],A,A).

rev(L,R):-ar(L,[],R).

palindrome(L):-rev(L,L).

## **Output:**



1. **Write a Prolog program to implement sumlist(L, S) so that S is the sum of a given list L.**

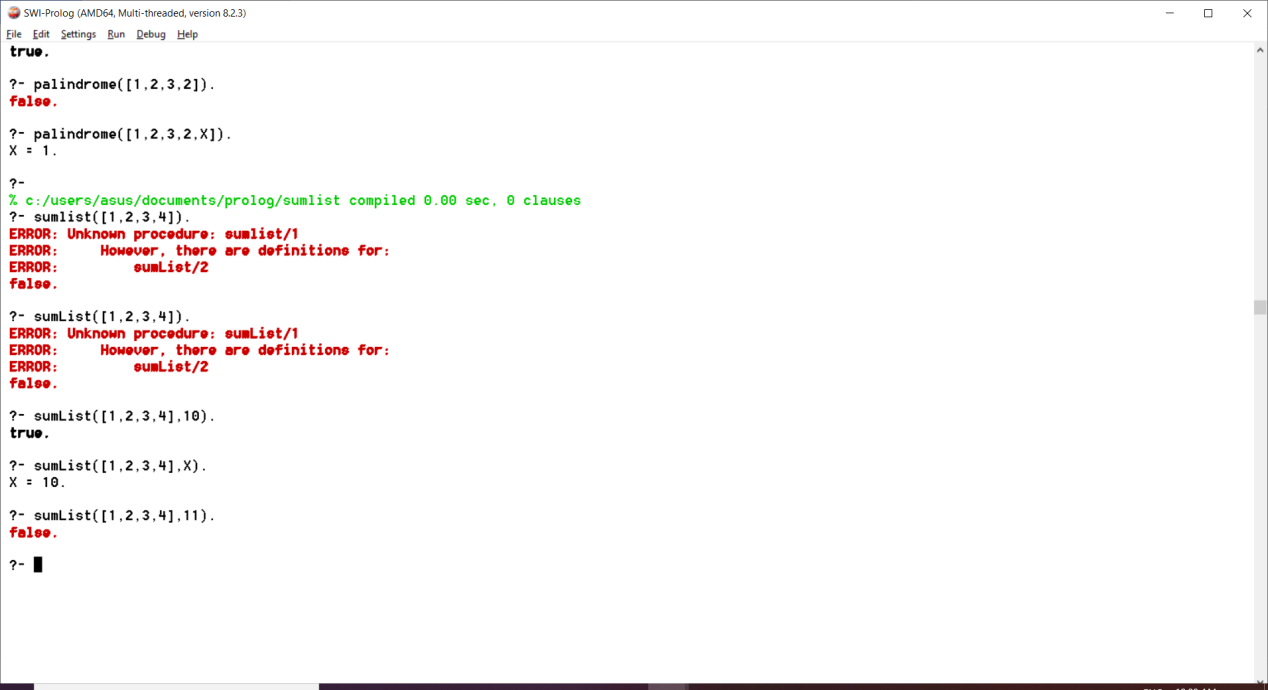
## **Program:**

sumList([],0).

sumList([H|T],S):-sumList(T,S1),

S is H + S1.

## **Output:**



1. **Write a Prolog program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.**

**Program:**

evenlength([]).

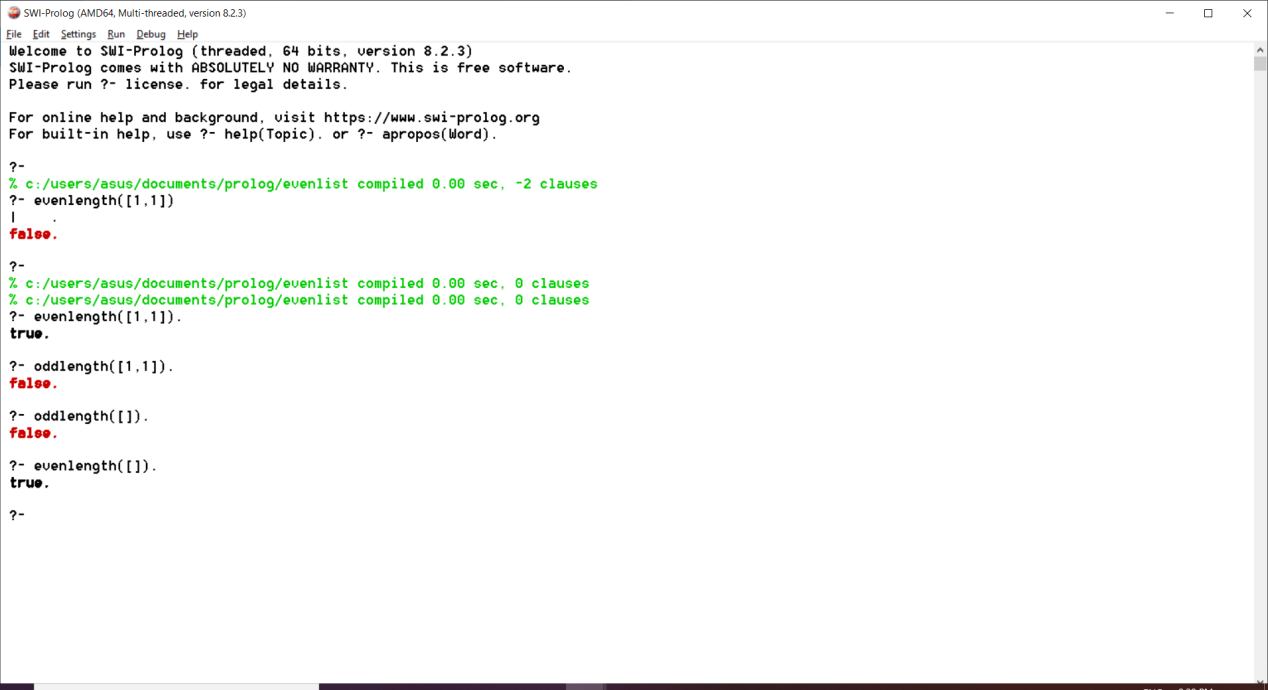
evenlength([\_|R]):-

oddlength(R).

oddlength([\_|R]):-

evenlength(R).

**Output:**



1. **Write a prolog program to implement nth\_element(N,L,X) where N is desired position in a list and X represents the Nth element of L.**

**Program:**

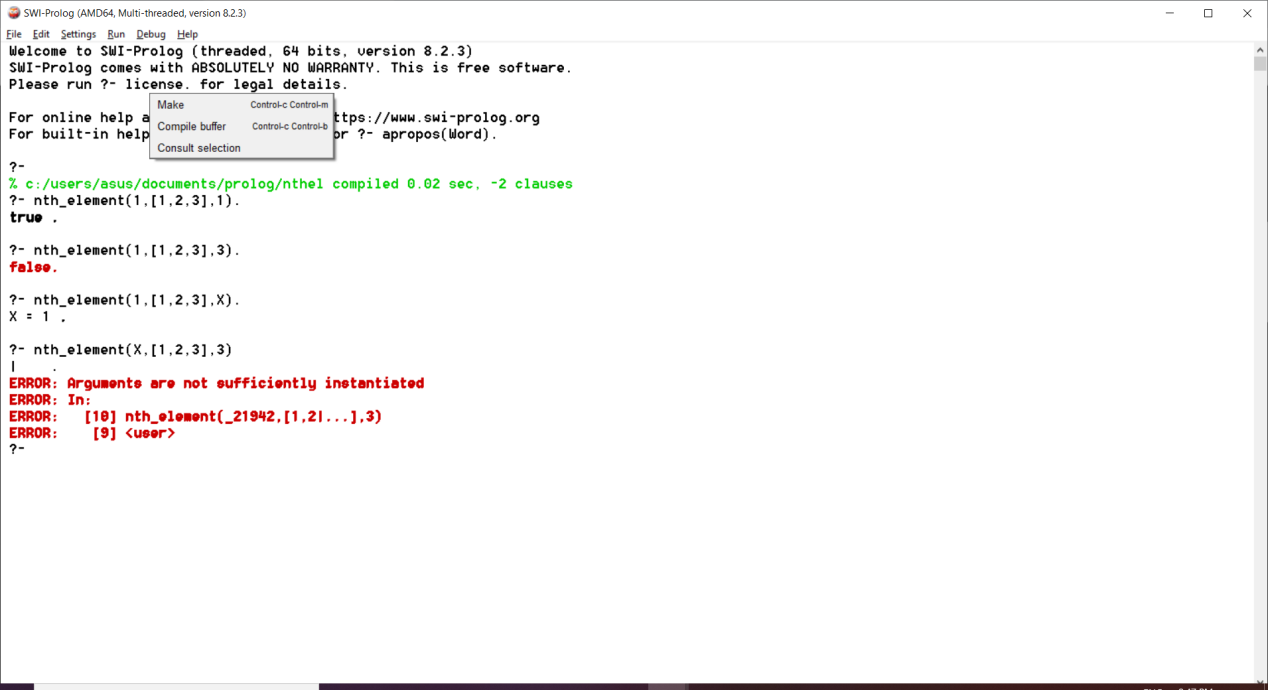
nth\_element(1,[H|\_],H).

nth\_element(N,[\_|T],X):-

N1 is N-1,

nth\_element(N1,T,X).

**Output:**



1. **Write a Prolog Program Implement remove\_dup(L,R) – L is list with some duplicates and R is list with removed duplicates.**

**Program:**

mem(X,[X|\_]).

mem(X,[\_|Y]):-mem(X,Y).

remove\_dup(L,M):-dupacc(L,[],M).

dupacc([],A,A).

dupacc([H|T],A,L):-

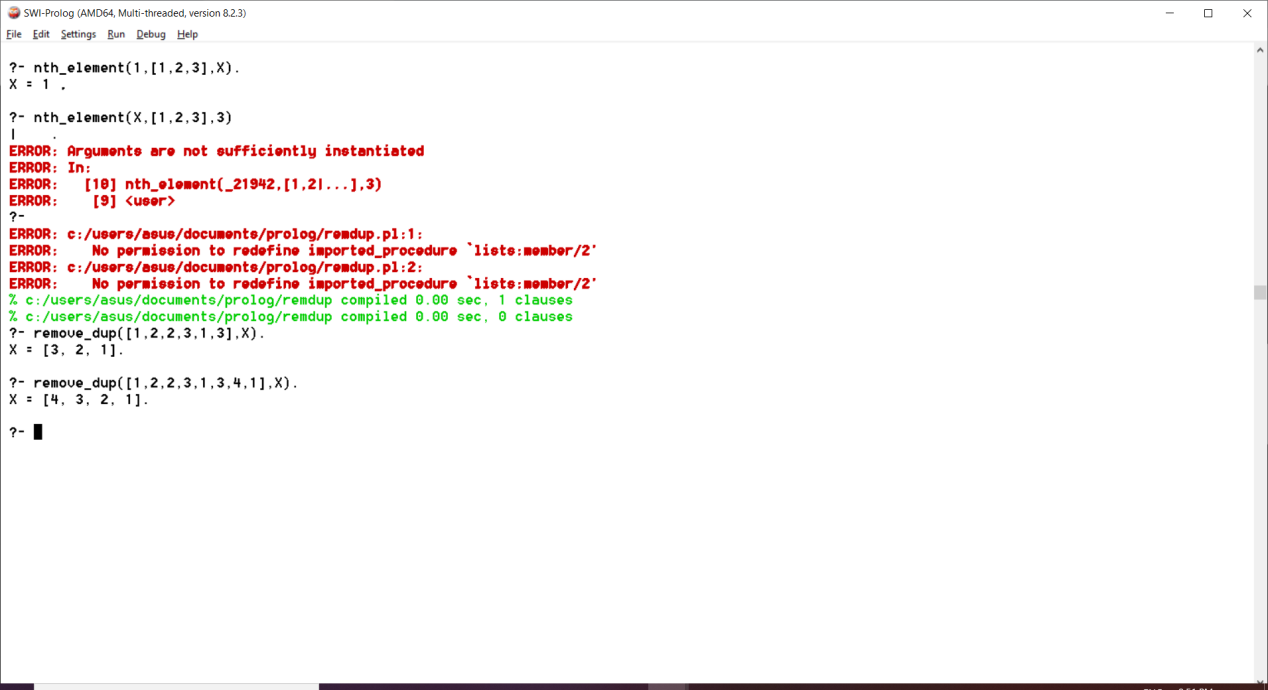
mem(H,A),

dupacc(T,A,L),

!.

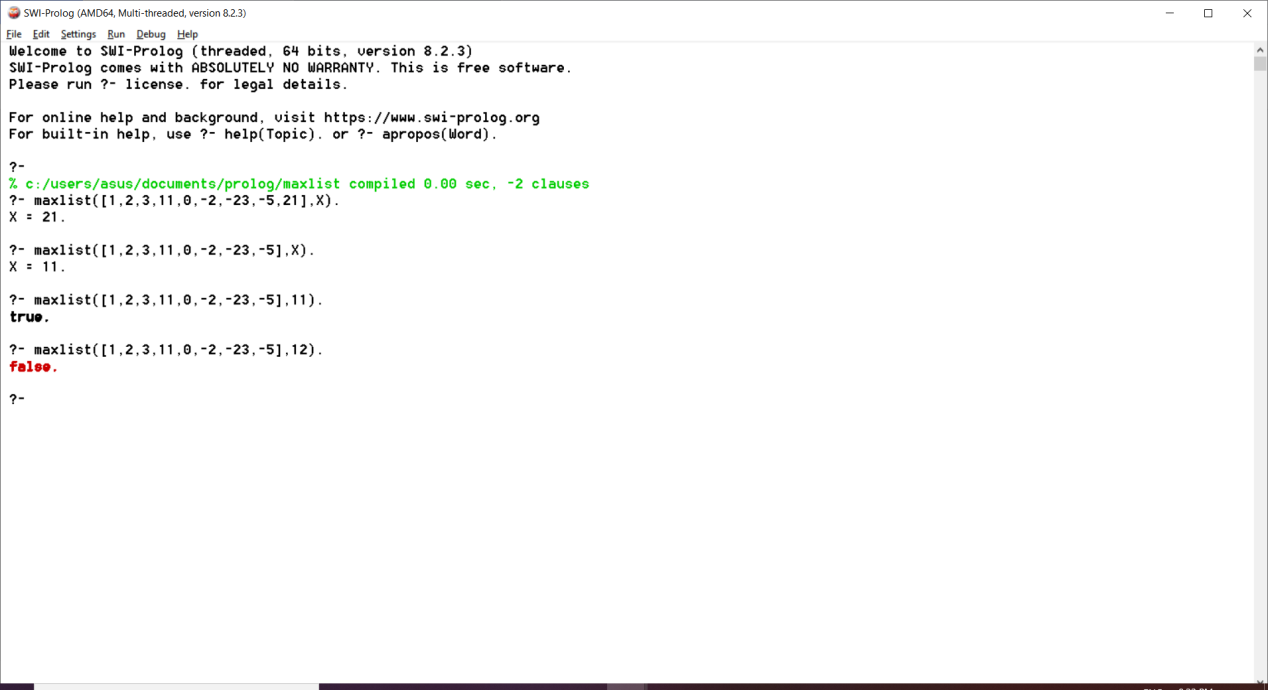
dupacc([H|T],A,L):-dupacc(T,[H|A],L).

**Output:**



1. **Write a Prolog program to implement maxlist(L, M) so that M is the maximum number in the list.**

**Program:** maxlist([H],H).

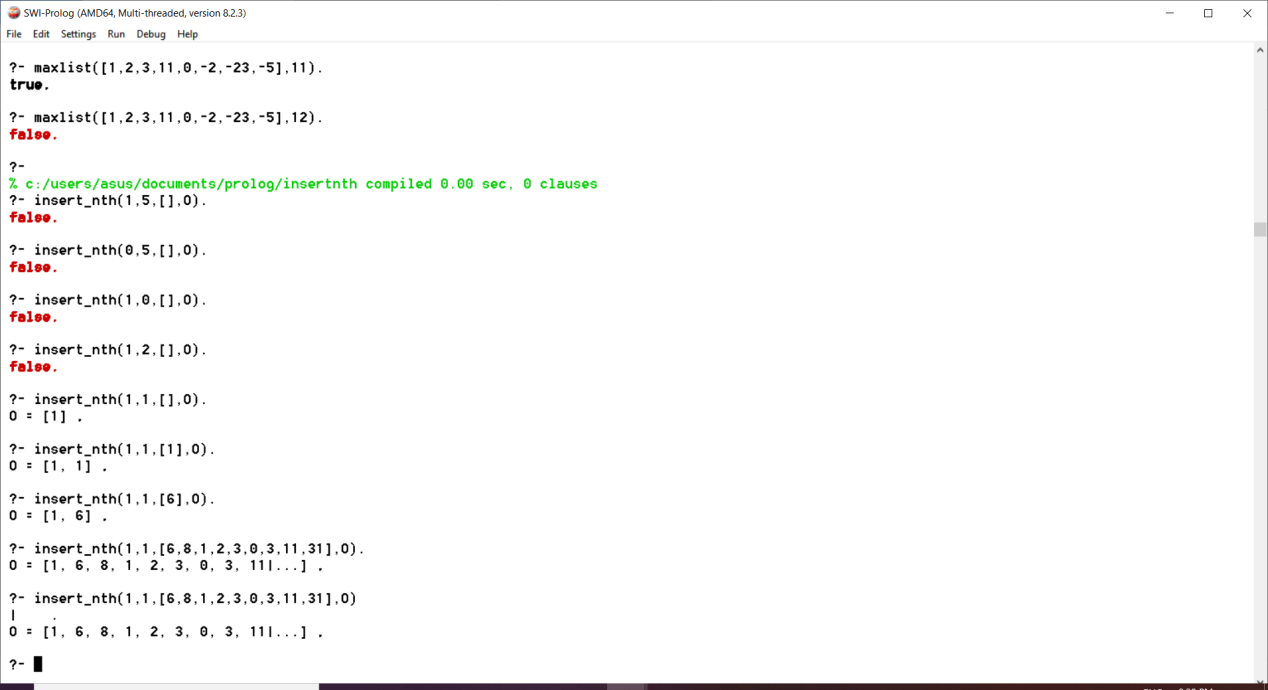
maxlist([H|T],M):- maxlist(T,M1), H<M1 -> M is M1;M is H.  
 **Output:**

1. **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.**

**Program:**

insert\_nth(I,1,L,[I|L]).

insert\_nth(I,N,[X|Y],[X|Z]):-A is N-1,insert\_nth(I,A,Y,Z).

**Output**:

1. **Write a program in prolog to implement sublists(S,L) that checks whether the List S is the sublist of List L or not.**

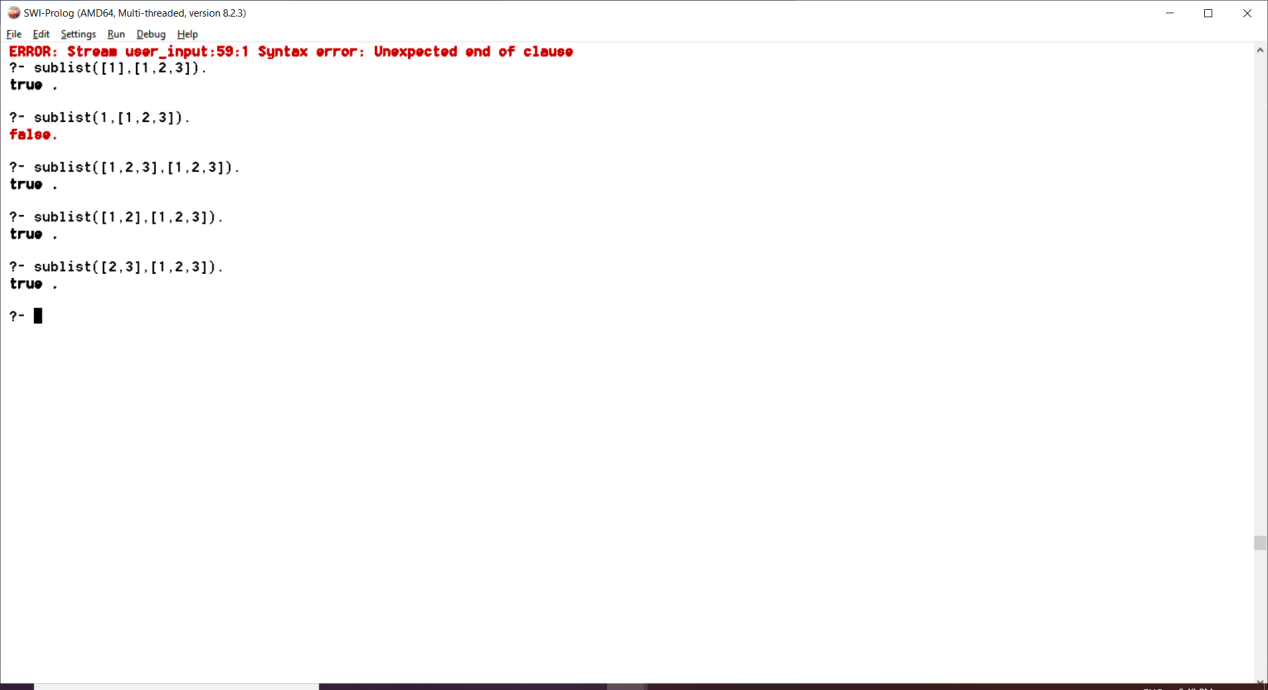
**Program:**

sublist([],\_).

sublist(\_,[]):- false.

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

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

**Output:**

1. **Write a prolog program to I****mplement delete\_nth(N,L,R) to remove Nth element from list L to generate list R.**

**Program:**

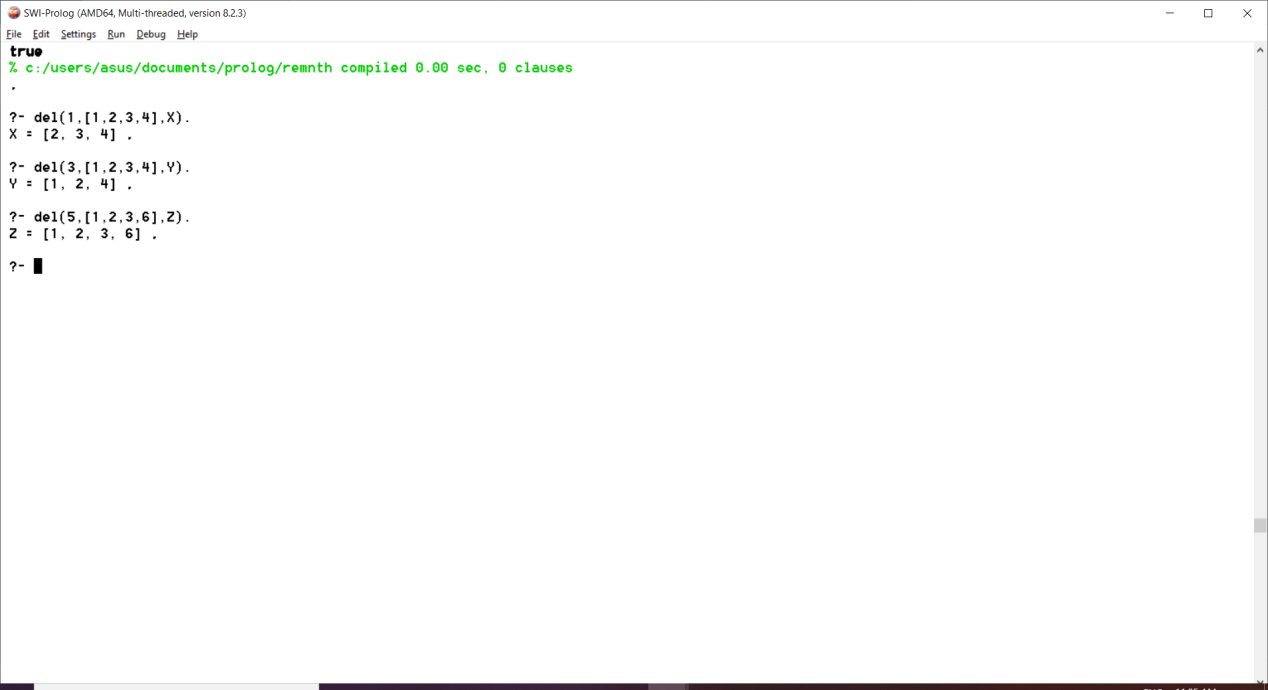
del(1,[\_|T],T).

del(\_,[],[]).

del(N,[H|T],[H|T1]):-

N1 is N-1,

del(N1,T,T1).

**Output:**

1. **Write a prolog program 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.**

**Program:**

delete\_all(\_, [], []).

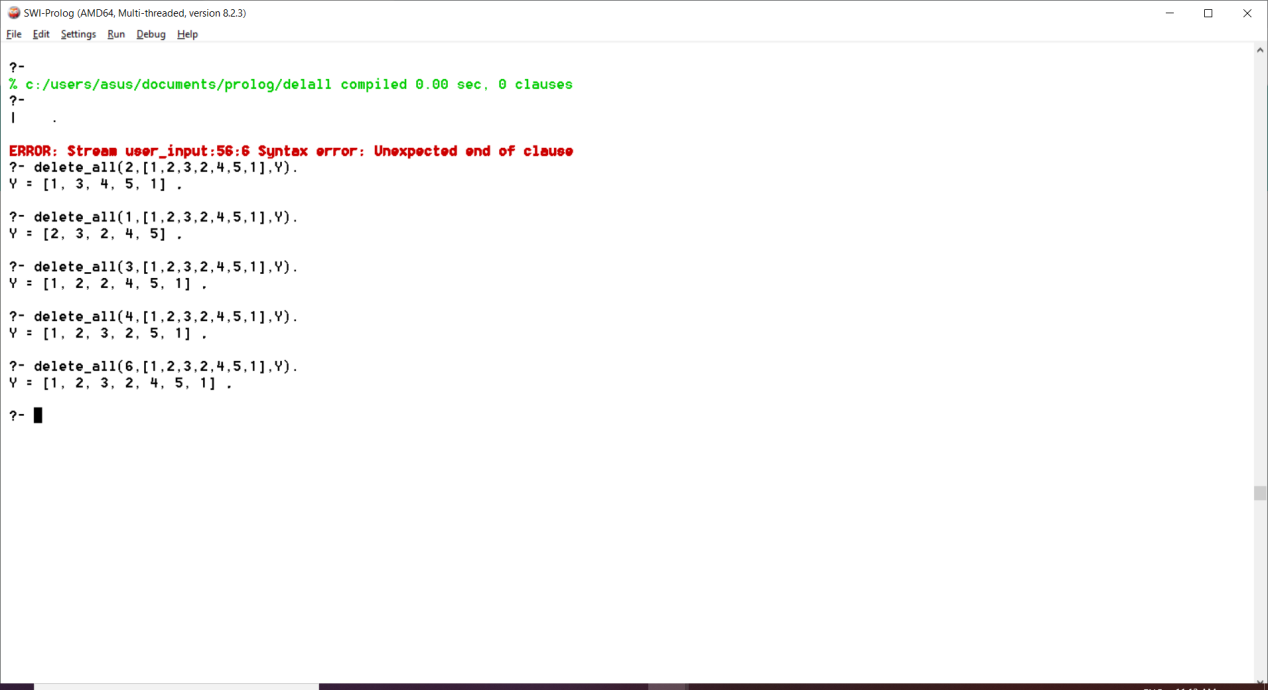
delete\_all(X, [X|T], R):-

delete\_all(X, T, R).

delete\_all(X, [H|T], [H|R]):-

delete\_all(X, T, R).

**Output:**

****

1. **Write a program in prolog to implement merge(L1,L2,L3) where L1 is first ordered list and L2 is a second order list and L3 represent the merged list.**

**Program:**

merge\_list([],[],[]).

merge\_list([],L2,L2).

merge\_list(L1,[],L1).

merge\_list([H1|T1],[H2|T2],[H1|T3]):-

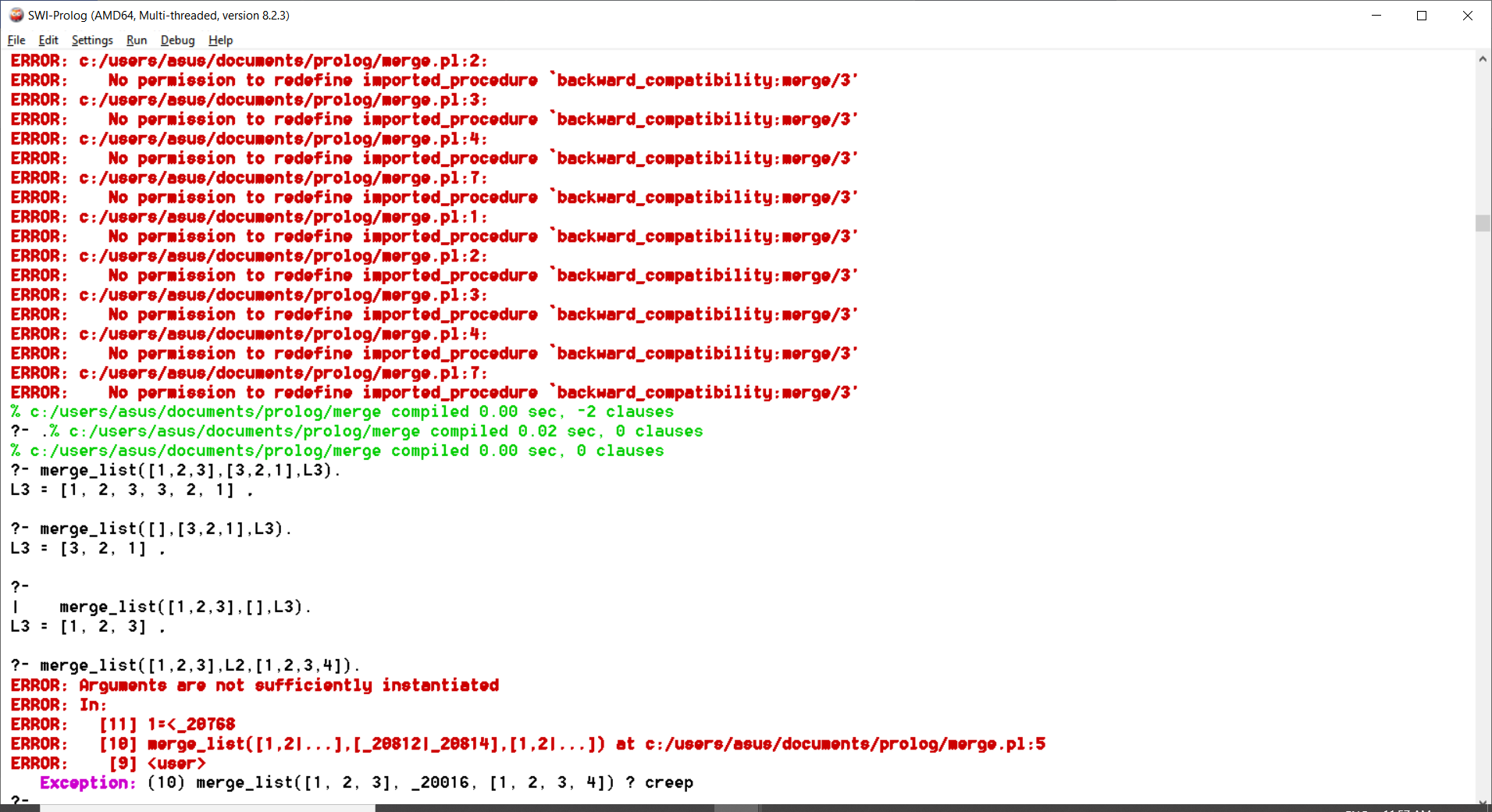
H1=<H2,

merge\_list(T1, [H2|T2], T3).

merge\_list([H1|T1],[H2|T2],[H2|T3]):-

merge\_list([H1|T1], T2, T3).

**Output:**



1. **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) :- !.**

**Program:**

sentence(A,C):-

nounphrase(A,B),

verbphrase(B,C).

nounphrase(A,C):-

article(A,B),

noun(B,C).

nounphrase(A,B):-

noun(A,B).

verbphrase(A,B):-

verb(A,B).

verbphrase(A,C):-

verb(A,B),

nounphrase(B,C).

verbphrase(A,C):-

verb(A,B),

prepositionphrase(B,C).

prepositionphrase(A,C):-

preposition(A,B),

nounphrase(B,C).

preposition([at|X],X).

article([the|X],X).

noun([dog|X],X).

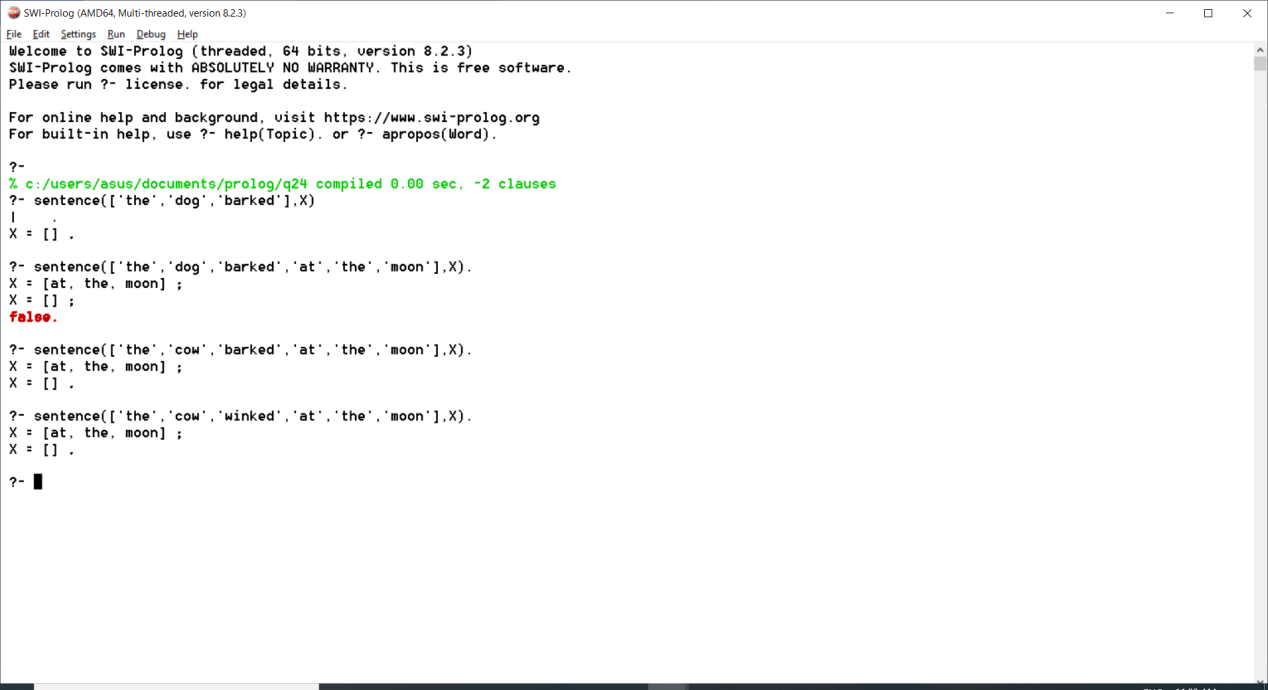
noun([cow|X],X).

noun([moon|X],X).

verb([barked|X],X).

verb([winked|X],X).

**Output:**



# **Write a prolog program that implements Semantic Networks (ATN/RTN).**

**Program:**

male(tom).

male(john).

male(albert).

female(alice).

female(victoria).

female(annie).

female(mary).

female(liz).

parents(edward,victoria).

parents(alice,albert).

parents(john,edward).

parents(bob,john).

parents(mary,john).

parents(tom,mary).

parents(charlie,alice).

parents(annie,alice).

parents(liz,charlie).

parents(edward,albert).

parents(alice,victoria).

offspring(X,Y):-parents(X,Y),!.

mother(X,Y):-female(X),parents(Y,X).

father(X,Y):-male(X),parents(Y,X).

sibling(X,Y):-parents(X,M),parents(Y,M).

sister\_of(X,Y):- female(X),parents(X,M),parents(Y,M),X\=Y.

aunt(X,Y):-sister\_of(X,F),parents(Y,F).

is\_son(Y):-male(Y),parents(Y,\_).

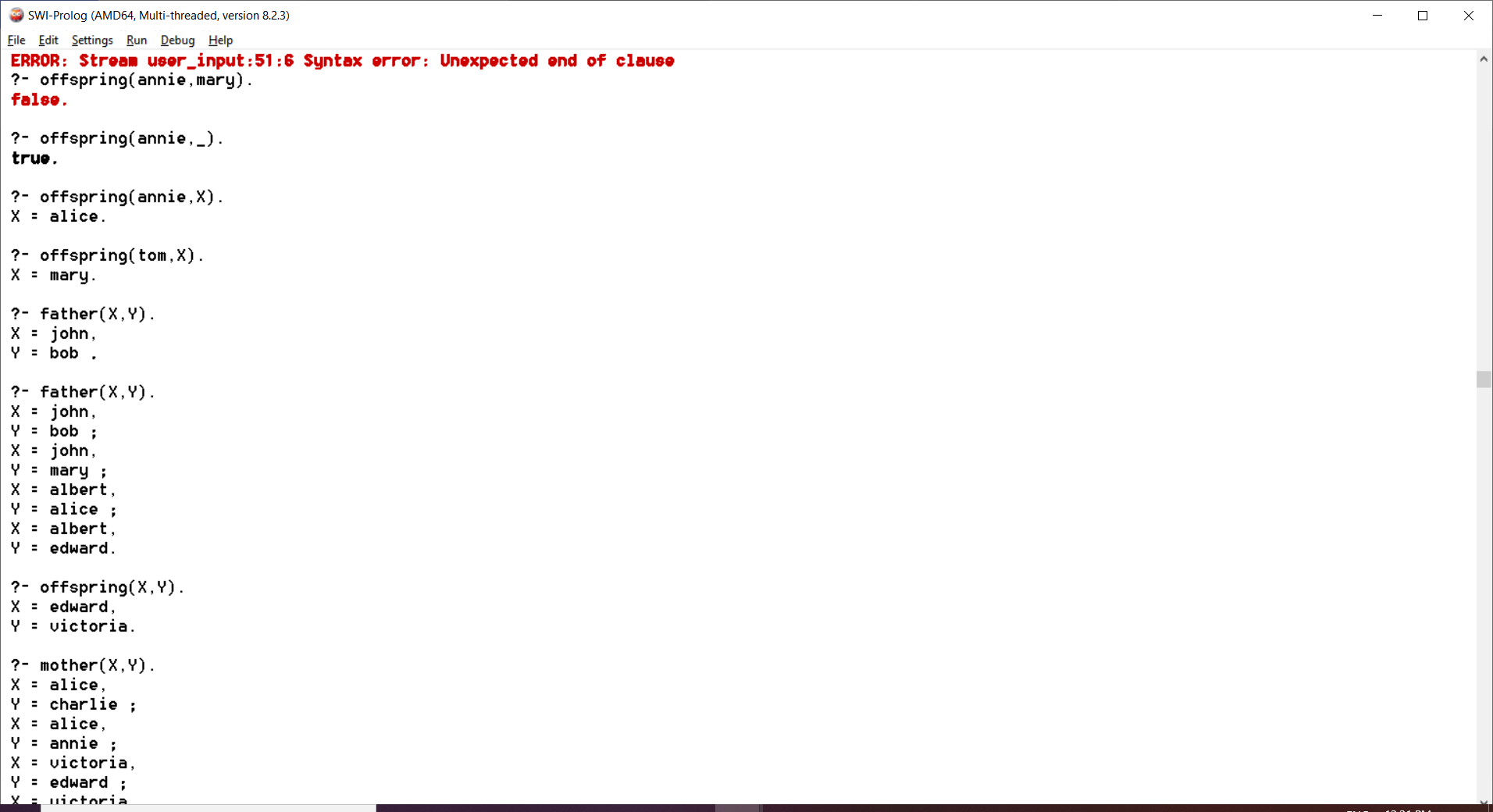
is\_mother(X):-female(X),mother(X,\_).

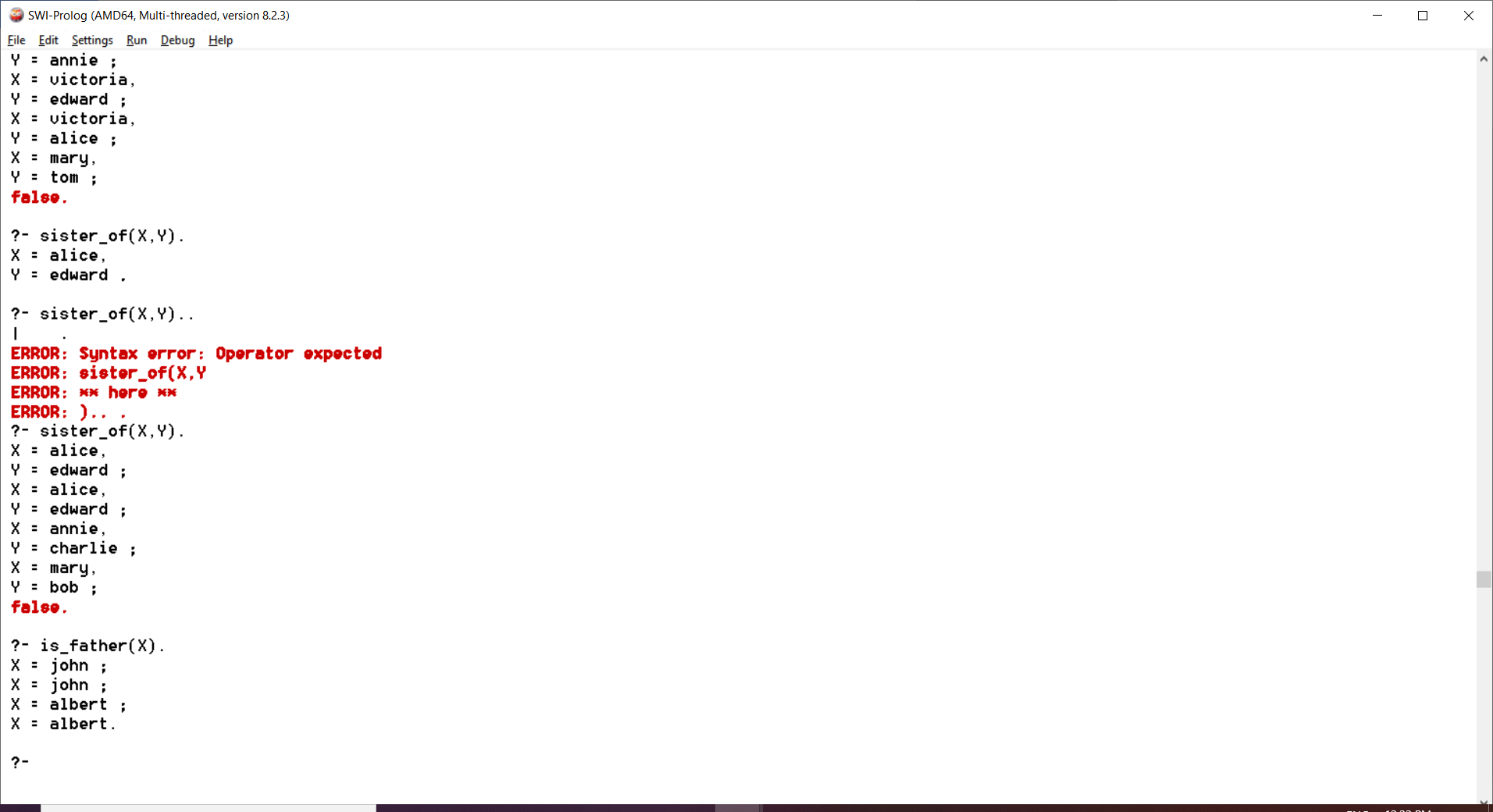
is\_father(X):-male(X),father(X,\_).

grandpa\_of(X,Y):-male(X),offspring(M,X),father(M,Y).

diff\_of(X,Y):-X\=Y.

**Output:**





**EXTRA QUESTION**

**Implement a family tree with following clauses:**

1. **is\_mother(X) % X is a mother**
2. **is\_father(X) % X is a father**
3. **is\_son(X) % X is a son**
4. **sibling(X,Y) % X is sibling of Y**
5. **sister\_of(X,Y)**
6. **child(X,Y)**
7. **mother(X,Y) % X is mother of Y**
8. **father(X,Y)**
9. **grandpa\_of(X,Y)**
10. **aunt(X,Y).**

**Program:**

female(victoria).

female(alice).

female(mary).

female(rosy).

female(jane).

male(albert).

male(edward).

male(harry).

male(mark).

parent(mary,harry,victoria).

parent(mary,harry,jane).

parent(rosy,mark,albert).

parent(victoria,albert,alice). %victoria is mother and albert is father of alice

parent(victoria,albert,edward).

sisterOf(X,Y):-

female(X),

parent(A,B,X),

parent(A,B,Y).

isMother(X):-

parent(X,\_,\_).

isFather(Y):-

parent(\_,Y,\_).

isSon(Z):-

male(Z),

parent(\_,\_,Z).

sibling(A,B):-

parent(X,Y,A),

parent(X,Y,B).

child(X,Y):-

parent(\_,Y,X). %Y is mother

child(X,Y):-

parent(Y,\_,X). %Y is father

motherOf(X,Y):-

parent(X,\_,Y).

fatherOf(X,Y):-

parent(\_,X,Y).

grandPaOf(X,Y):-

male(X),

child(Y,Z),

child(Z,X).

%paternal grandpa

auntOf(X,Y):-

motherOf(Z,Y),

sisterOf(X,Z). %maternal aunt

auntOf(X,Y):-

fatherOf(Z,Y),

sisterOf(X,Z). %paternal aunt

**Output:**

