**DAY-2 LAB PROGRAMS**

S.Kangana Sri

192210006

CSA1760-Artificial Intelligence

**19.Student Teacher**

% Facts: student(Name, ID)

student('Alice', 101).

student('Bob', 102).

student('Charlie', 103).

student('Diana', 104).

% Facts: teacher(Name, ID)

teacher('Dr. Smith', 201).

teacher('Prof. Johnson', 202).

teacher('Dr. Lee', 203).

% Facts: subject(SubjectName, SubjectCode, TeacherID)

subject('Mathematics', 'MTH101', 201).

subject('Physics', 'PHY102', 202).

subject('Computer Science', 'CS103', 203).

subject('Chemistry', 'CHE104', 202).

% Facts: student\_enrolled\_in(StudentID, SubjectCode)

student\_enrolled\_in(101, 'MTH101').

student\_enrolled\_in(101, 'CS103').

student\_enrolled\_in(102, 'PHY102').

student\_enrolled\_in(103, 'CS103').

student\_enrolled\_in(104, 'CHE104').

% Rule: Find subjects taught by a teacher

find\_subjects\_by\_teacher(TeacherName, SubjectName, SubjectCode) :-

teacher(TeacherName, TeacherID),

subject(SubjectName, SubjectCode, TeacherID),

write('Teacher: '), write(TeacherName), nl,

write('Subject: '), write(SubjectName), nl,

write('Subject Code: '), write(SubjectCode), nl.

% Rule: Find students enrolled in a specific subject

find\_students\_in\_subject(SubjectCode, StudentName, StudentID) :-

subject(SubjectName, SubjectCode, TeacherID),

student\_enrolled\_in(StudentID, SubjectCode),

student(StudentName, StudentID),

write('Student: '), write(StudentName), nl,

write('Subject: '), write(SubjectName), nl,

write('Subject Code: '), write(SubjectCode), nl.

% Rule: Find teachers who teach a specific subject

find\_teacher\_for\_subject(SubjectCode, TeacherName) :-

subject(SubjectName, SubjectCode, TeacherID),

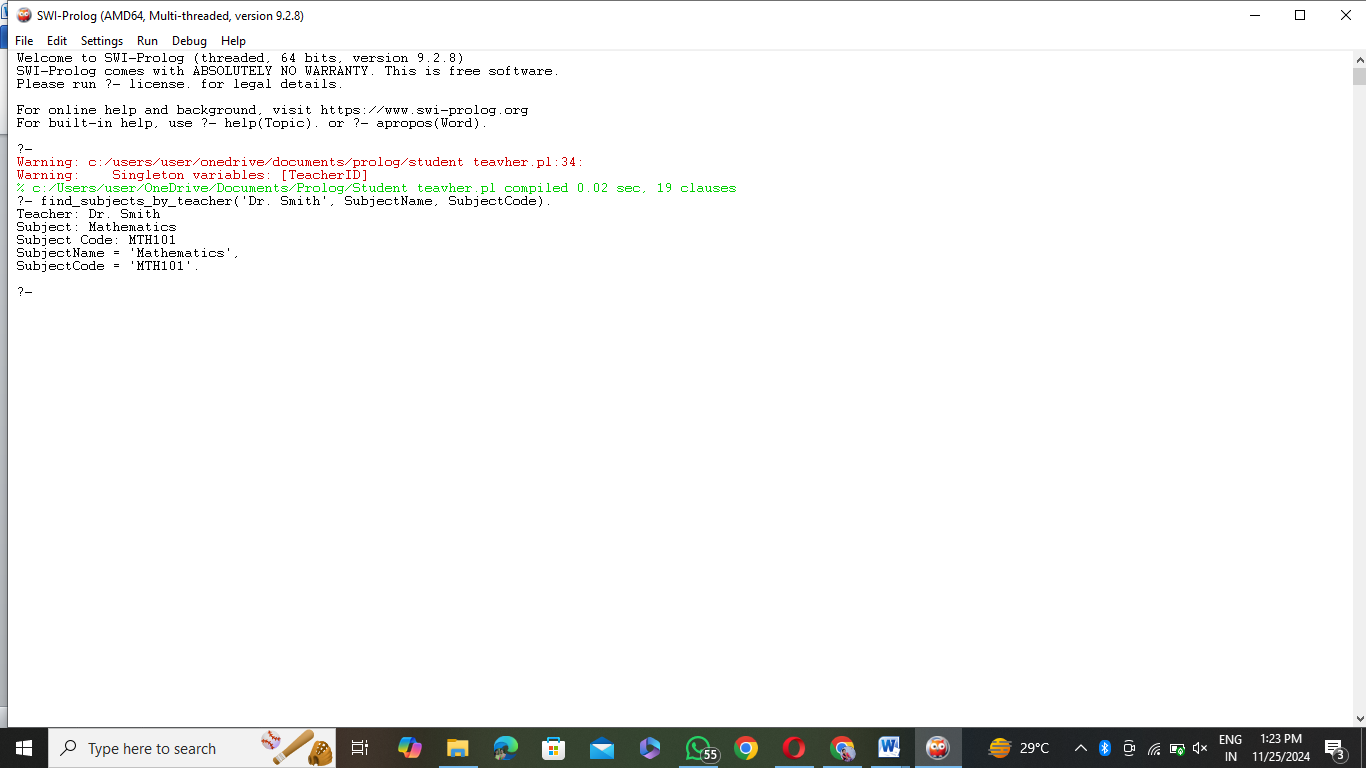
teacher(TeacherName, TeacherID),

write('Subject: '), write(SubjectName), nl,

write('Subject Code: '), write(SubjectCode), nl,

write('Teacher: '), write(TeacherName), nl.

**Output:**



**20.Plants Database:**  
% Facts: planet(Name, Type, DistanceFromSun, HasMoons)

planet('Mercury', rocky, 57.9, no).

planet('Venus', rocky, 108.2, no).

planet('Earth', rocky, 149.6, yes).

planet('Mars', rocky, 227.9, yes).

planet('Jupiter', gas\_giant, 778.3, yes).

planet('Saturn', gas\_giant, 1427, yes).

planet('Uranus', ice\_giant, 2871, yes).

planet('Neptune', ice\_giant, 4497.1, yes).

% Facts: moon(MoonName, PlanetName)

moon('Moon', 'Earth').

moon('Phobos', 'Mars').

moon('Deimos', 'Mars').

moon('Io', 'Jupiter').

moon('Europa', 'Jupiter').

moon('Ganymede', 'Jupiter').

moon('Titan', 'Saturn').

moon('Rhea', 'Saturn').

moon('Triton', 'Neptune').

% Rule: Find the planet type by its name

find\_planet\_type(PlanetName, Type) :-

planet(PlanetName, Type, \_, \_),

write('Planet: '), write(PlanetName), nl,

write('Type: '), write(Type), nl.

% Rule: Find the distance from the Sun of a planet

find\_planet\_distance(PlanetName, Distance) :-

planet(PlanetName, \_, Distance, \_),

write('Planet: '), write(PlanetName), nl,

write('Distance from Sun: '), write(Distance), write(' million km'), nl.

% Rule: Find whether a planet has moons

find\_planet\_moons(PlanetName, HasMoons) :-

planet(PlanetName, \_, \_, HasMoons),

write('Planet: '), write(PlanetName), nl,

write('Has Moons: '), write(HasMoons), nl.

% Rule: Find moons of a planet

find\_moons\_of\_planet(PlanetName) :-

moon(MoonName, PlanetName),

write('Moon of '), write(PlanetName), write(': '), write(MoonName), nl,

fail.

find\_moons\_of\_planet(PlanetName) :-

write('No more moons for planet: '), write(PlanetName), nl.

% Rule: Find planets that are gas giants

find\_gas\_giants :-

planet(PlanetName, gas\_giant, \_, \_),

write('Gas Giant: '), write(PlanetName), nl,

fail.

find\_gas\_giants :-

write('End of gas giants list.'), nl.

% Rule: Find planets that are rocky

find\_rocky\_planets :-

planet(PlanetName, rocky, \_, \_),

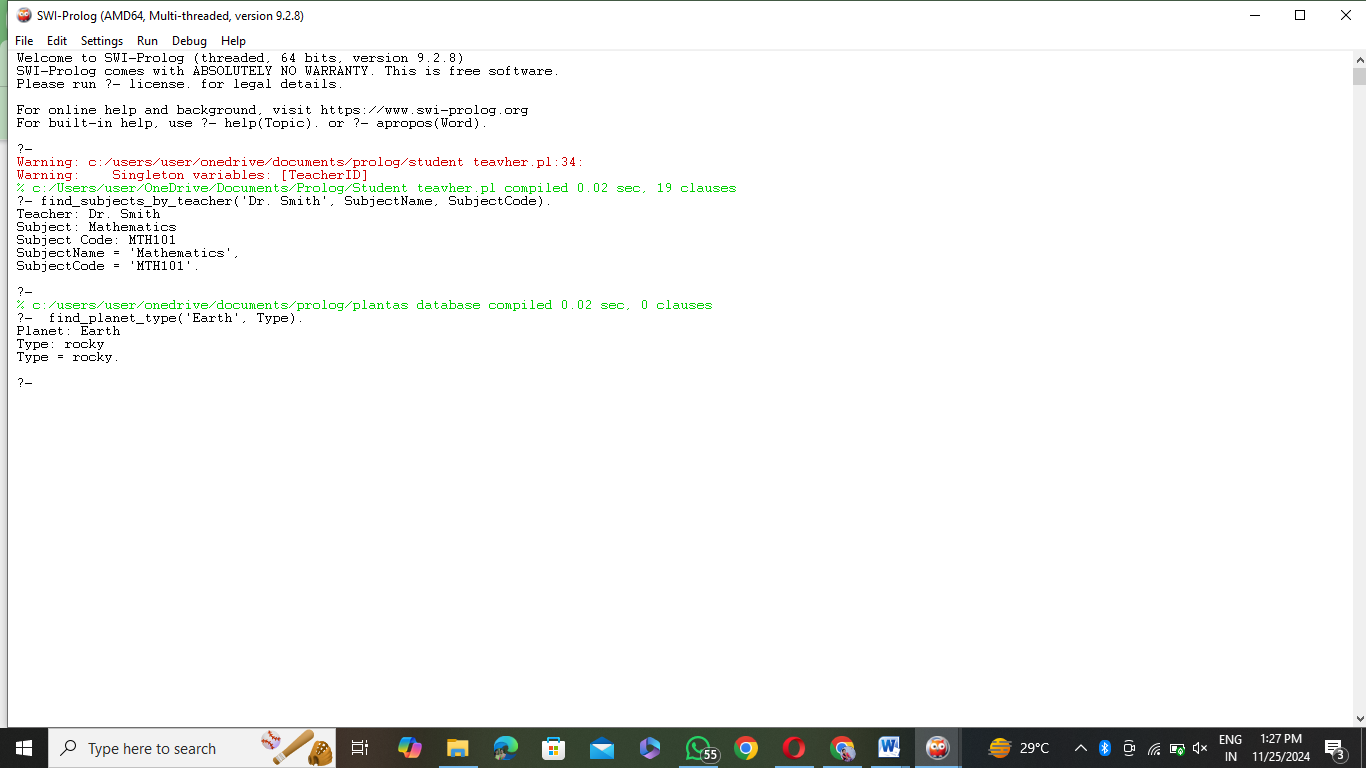
write('Rocky Planet: '), write(PlanetName), nl,

fail.

find\_rocky\_planets :-

write('End of rocky planets list.'), nl.

**Output:**

****

**21.Towers of Hanai**

% Base case: No disks to move, so nothing to do.

hanoi(0, \_, \_, \_) :-

write('All disks are already moved!'), nl, !.

% Recursive case: Move N disks from Source to Target using Auxiliary as helper peg.

hanoi(N, Source, Target, Auxiliary) :-

N > 0,

N1 is N - 1,

% Move N-1 disks from Source to Auxiliary using Target as helper

hanoi(N1, Source, Auxiliary, Target),

% Move the N-th disk from Source to Target

write('Move disk '), write(N), write(' from '), write(Source), write(' to '), write(Target), nl,

% Move the N-1 disks from Auxiliary to Target using Source as helper

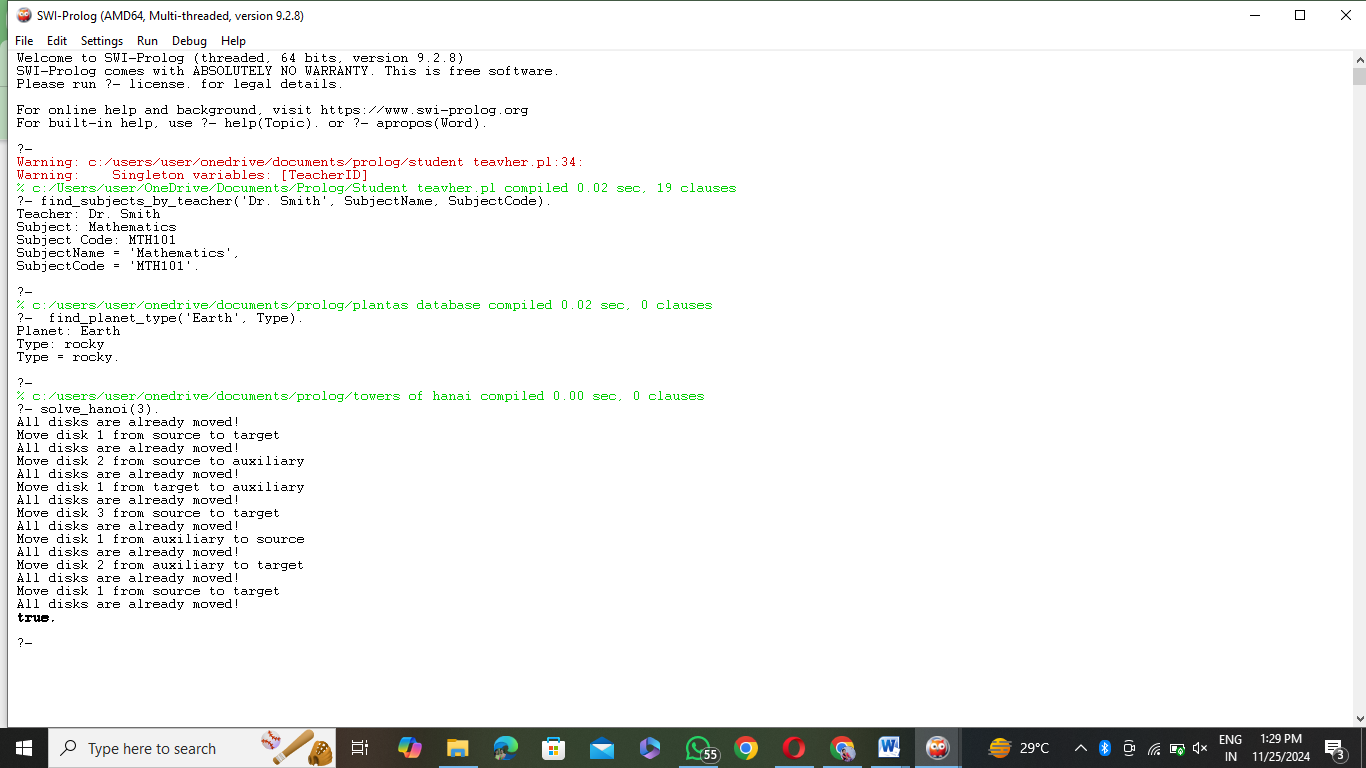
hanoi(N1, Auxiliary, Target, Source).

% To start the solution for N disks, call this predicate:

solve\_hanoi(N) :-

hanoi(N, source, target, auxiliary).

**Output:**

****

**22.Bird can fly**

% Facts: bird(Name, CanFly)

% Name: The name of the bird.

% CanFly: Whether the bird can fly (yes or no).

bird('Eagle', yes).

bird('Sparrow', yes).

bird('Penguin', no).

bird('Ostrich', no).

bird('Bat', yes). % Bat is technically a mammal, but it can fly, so added for fun.

% Rule: Can a bird fly?

can\_fly(BirdName) :-

bird(BirdName, yes),

write(BirdName), write(' can fly!'), nl.

can\_fly(BirdName) :-

bird(BirdName, no),

write(BirdName), write(' cannot fly.'), nl.

% Rule: Birds that cannot fly (for example)

cannot\_fly(BirdName) :-

bird(BirdName, no),

write(BirdName), write(' cannot fly.'), nl.

% Rule: Birds that can fly (for example)

can\_fly\_all :-

bird(BirdName, yes),

write(BirdName), write(' can fly!'), nl,

fail.

can\_fly\_all :-

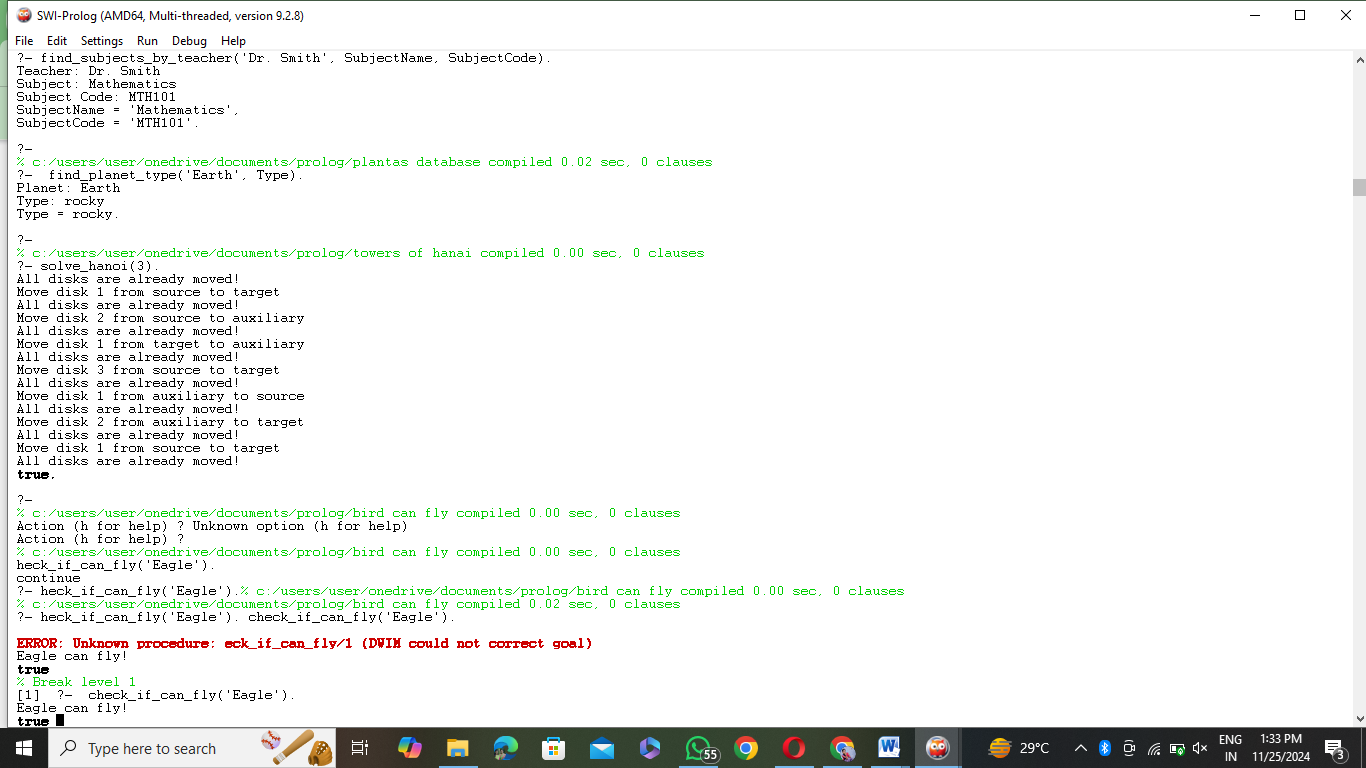
write('End of bird list that can fly.'), nl.

% Entry point: Check if a specific bird can fly.

check\_if\_can\_fly(BirdName) :-

can\_fly(BirdName).

**Output:**

****

**23.Family Tree**

% Facts: parent(Parent, Child)

% Parent is a parent of the Child.

parent(john, mary).

parent(john, james).

parent(mary, lisa).

parent(mary, tom).

parent(james, sam).

parent(susan, james).

parent(lisa, kate).

parent(tom, emily).

% Facts: gender(Name, Gender)

% Gender of the person.

gender(john, male).

gender(mary, female).

gender(james, male).

gender(susan, female).

gender(lisa, female).

gender(tom, male).

gender(kate, female).

gender(sam, male).

gender(emily, female).

% Rule: siblings(Child1, Child2)

% Two children are siblings if they have the same parent.

siblings(Child1, Child2) :-

parent(Parent, Child1),

parent(Parent, Child2),

Child1 \= Child2.

% Rule: father(Father, Child)

% A father is a male parent.

father(Father, Child) :-

parent(Father, Child),

gender(Father, male).

% Rule: mother(Mother, Child)

% A mother is a female parent.

mother(Mother, Child) :-

parent(Mother, Child),

gender(Mother, female).

% Rule: grandparent(Grandparent, Grandchild)

% A grandparent is a parent of a parent.

grandparent(Grandparent, Grandchild) :-

parent(Grandparent, Parent),

parent(Parent, Grandchild).

% Rule: grandfather(Grandfather, Grandchild)

% A grandfather is a male grandparent.

grandfather(Grandfather, Grandchild) :-

grandparent(Grandfather, Grandchild),

gender(Grandfather, male).

% Rule: grandmother(Grandmother, Grandchild)

% A grandmother is a female grandparent.

grandmother(Grandmother, Grandchild) :-

grandparent(Grandmother, Grandchild),

gender(Grandmother, female).

% Rule: cousin(Cousin1, Cousin2)

% Two persons are cousins if their parents are siblings.

cousin(Cousin1, Cousin2) :-

parent(Parent1, Cousin1),

parent(Parent2, Cousin2),

siblings(Parent1, Parent2).

% Rule: aunt(Aunt, NieceOrNephew)

% An aunt is a female sibling of a parent.

aunt(Aunt, NieceOrNephew) :-

parent(Parent, NieceOrNephew),

siblings(Parent, Aunt),

gender(Aunt, female).

% Rule: uncle(Uncle, NieceOrNephew)

% An uncle is a male sibling of a parent.

uncle(Uncle, NieceOrNephew) :-

parent(Parent, NieceOrNephew),

siblings(Parent, Uncle),

gender(Uncle, male).

% Query examples for family relationships

% Find all siblings of Lisa

% ?- siblings(lisa, Sibling).

% Find all children of John

% ?- parent(john, Child).

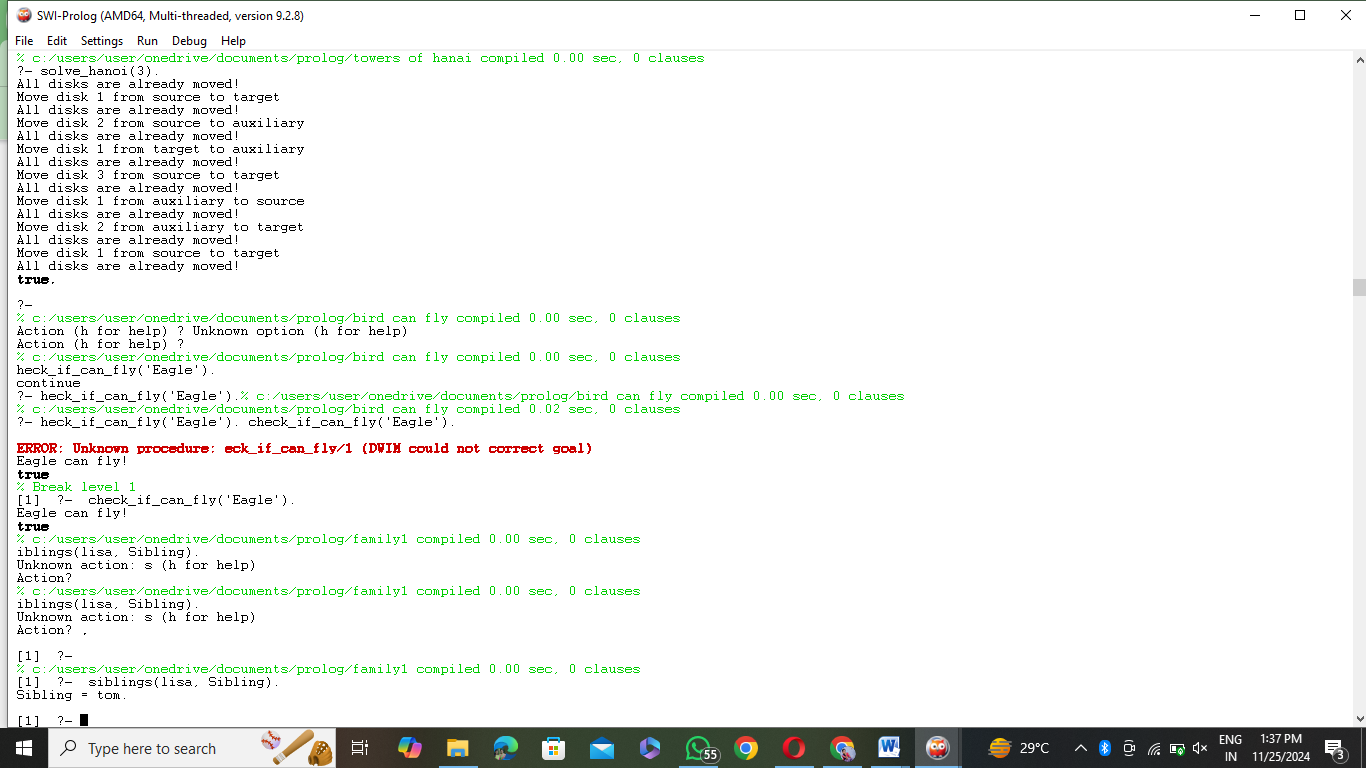
% Find all grandparents of Kate

% ?- grandparent(Grandparent, kate).

% Find all cousins of Sam

% ?- cousin(sam, Cousin).

**Output:**

****

**24.Dieting System:**

% Facts: disease(Disease, Symptoms)

% Defines diseases and their symptoms

disease(diabetes, [high\_blood\_sugar, frequent\_urination, fatigue]).

disease(hypertension, [high\_blood\_pressure, headache, dizziness]).

disease(anemia, [fatigue, weakness, pale\_skin]).

disease(obesity, [high\_body\_mass\_index, fatigue, breathlessness]).

disease(gastritis, [stomach\_pain, nausea, bloating]).

% Facts: diet(Disease, Foods)

% Defines recommended diets for diseases

diet(diabetes, [vegetables, lean\_meat, whole\_grains, low\_glycemic\_fruits]).

diet(hypertension, [low\_sodium\_foods, leafy\_greens, berries, oats]).

diet(anemia, [iron\_rich\_foods, leafy\_greens, red\_meat, beans, nuts]).

diet(obesity, [low\_calorie\_foods, vegetables, lean\_proteins, fruits]).

diet(gastritis, [low\_acidic\_foods, bananas, rice, boiled\_potatoes]).

% Facts: avoid(Disease, Foods)

% Defines foods to avoid for diseases

avoid(diabetes, [sugary\_snacks, white\_bread, soda, pastries]).

avoid(hypertension, [salty\_foods, processed\_meats, caffeine]).

avoid(anemia, [calcium\_rich\_foods\_with\_meals, tea, coffee]).

avoid(obesity, [high\_calorie\_snacks, fried\_foods, sugary\_drinks]).

avoid(gastritis, [spicy\_foods, alcohol, coffee, citrus\_fruits]).

% Rule: recommend\_diet(Disease, RecommendedFoods, AvoidFoods)

% Suggests a diet based on the disease

recommend\_diet(Disease, RecommendedFoods, AvoidFoods) :-

diet(Disease, RecommendedFoods),

avoid(Disease, AvoidFoods).

% Query Examples:

% 1. Suggest a diet for diabetes:

% ?- recommend\_diet(diabetes, RecommendedFoods, AvoidFoods).

% 2. List diseases associated with fatigue:

% ?- disease(Disease, Symptoms), member(fatigue, Symptoms).

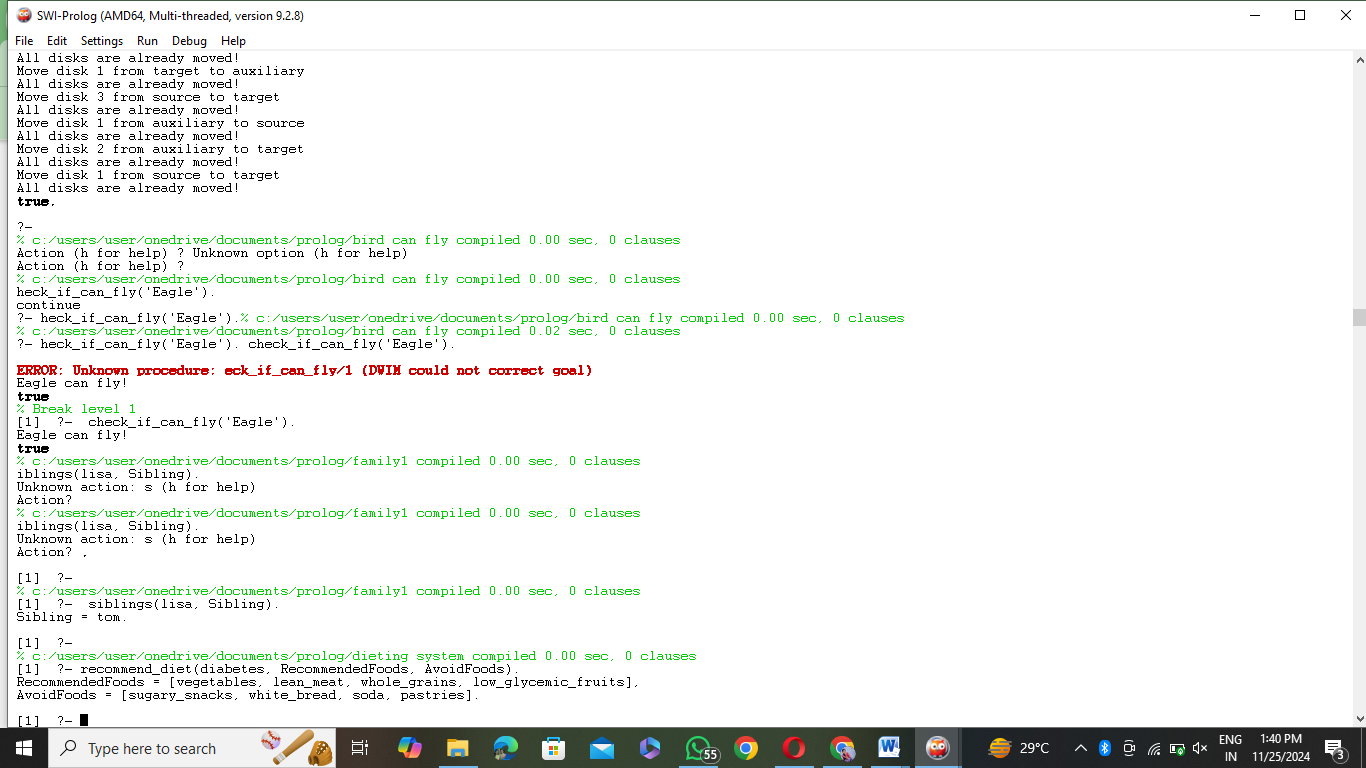
% 3. Check which foods to avoid for gastritis:

% ?- avoid(gastritis, AvoidFoods).

% 4. What foods are recommended for anemia:

% ?- diet(anemia, RecommendedFoods).

**Output:**

****