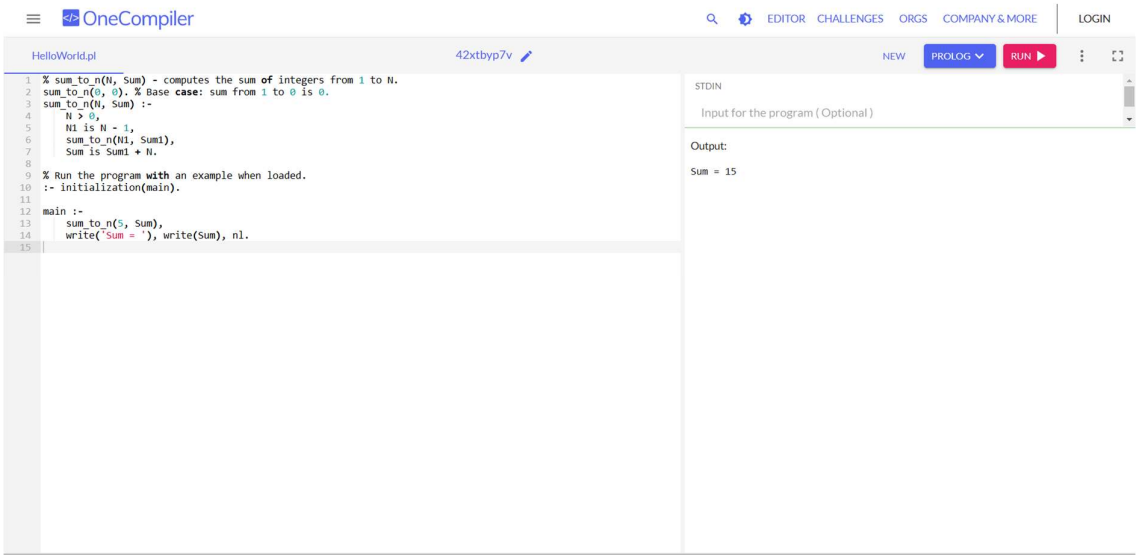
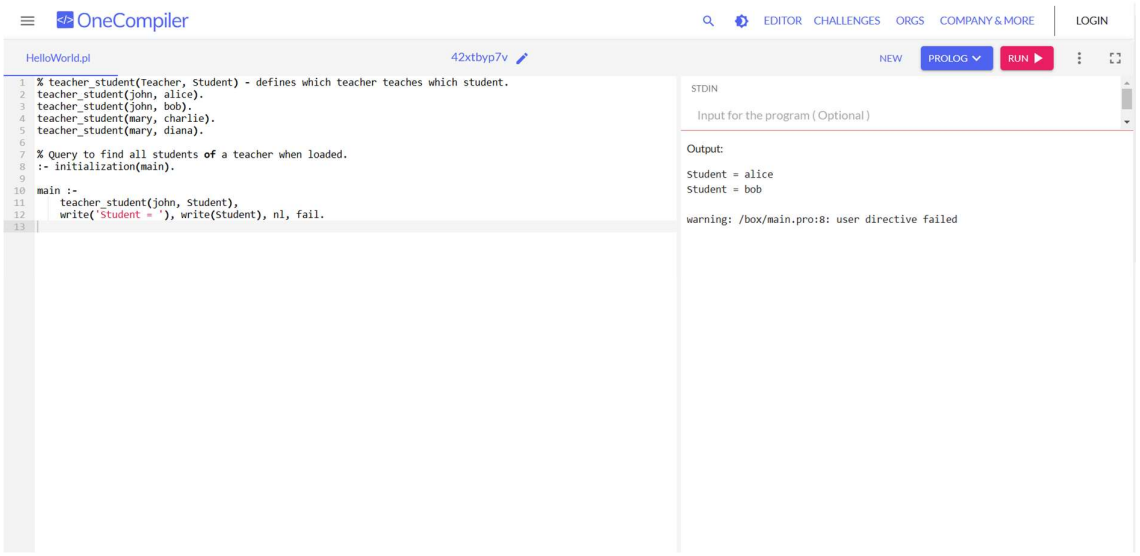
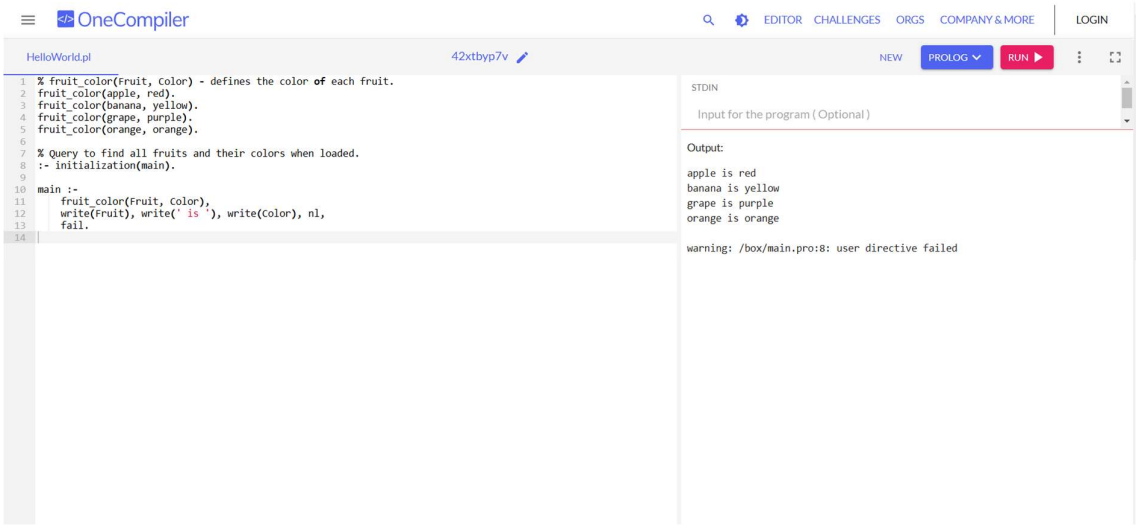
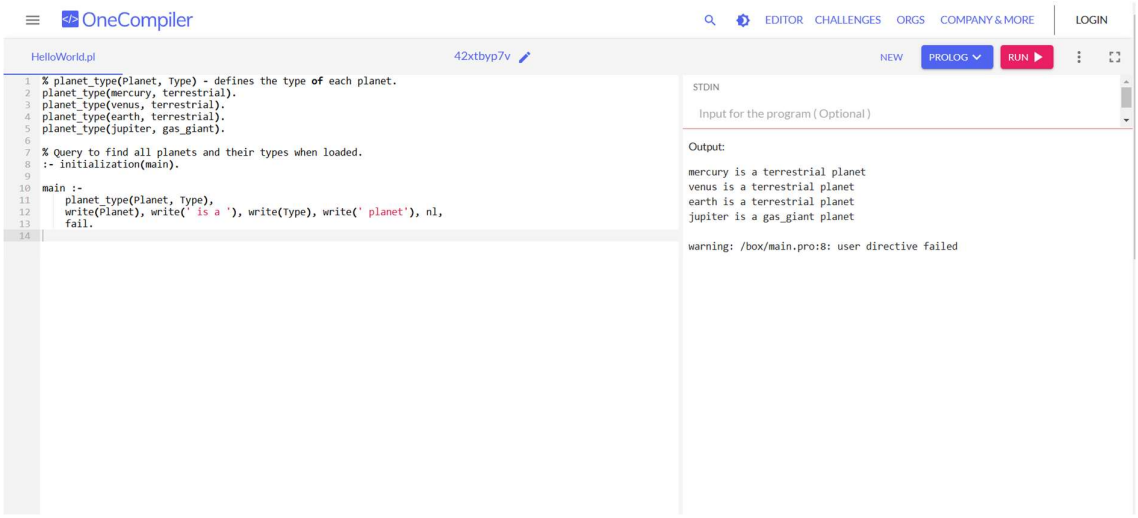
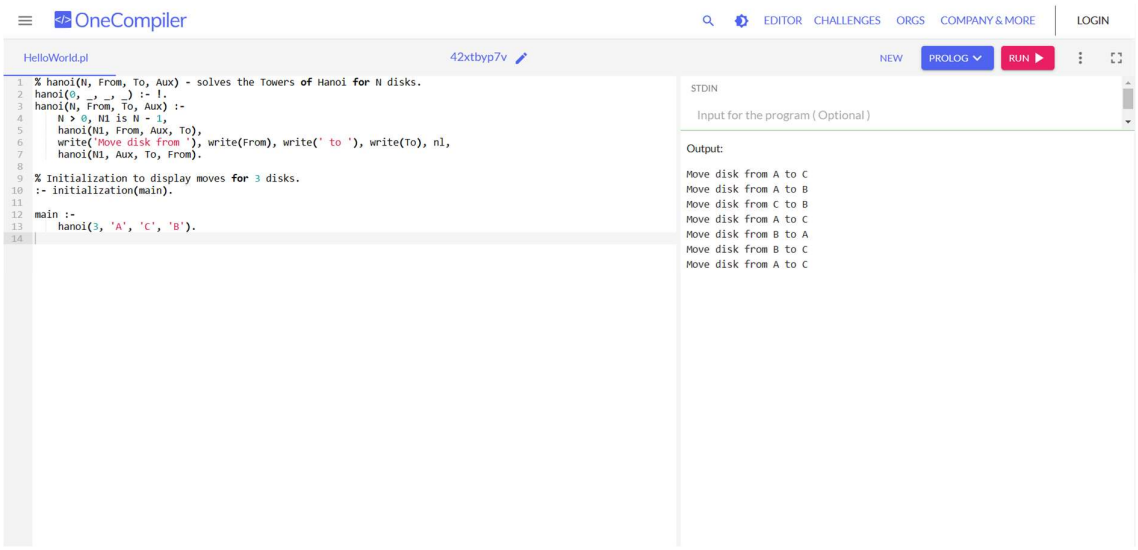
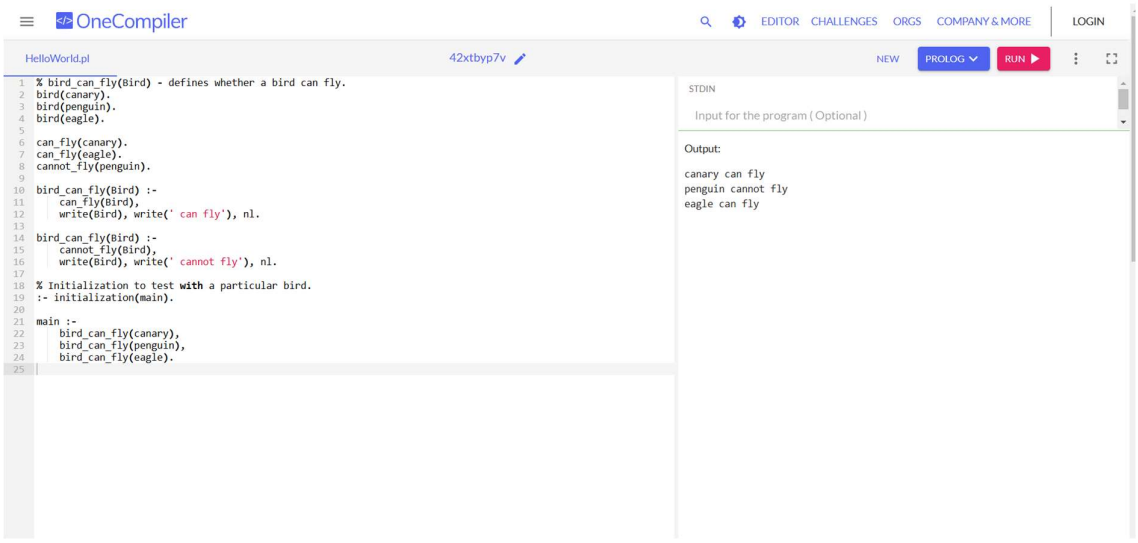

OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% sum_to_n(N, Sum) - computes the sum of integers from 1 to N.
sum_to_n(0, 0). % Base case: sum from 1 to 0 is 0.
sum_to_n(N, Sum) :-
N > 0,
N1 is N - 1,
sum_to_n(N1, Sum1),
Sum is Sum1 + N.
% Run the program with an example when loaded.
:- initialization(main).
main :-
sum_to_n(5, Sum),
write('Sum = '), write(Sum), nl.
STDIN
Input for the program (Optional)
Output:
Sum = 15

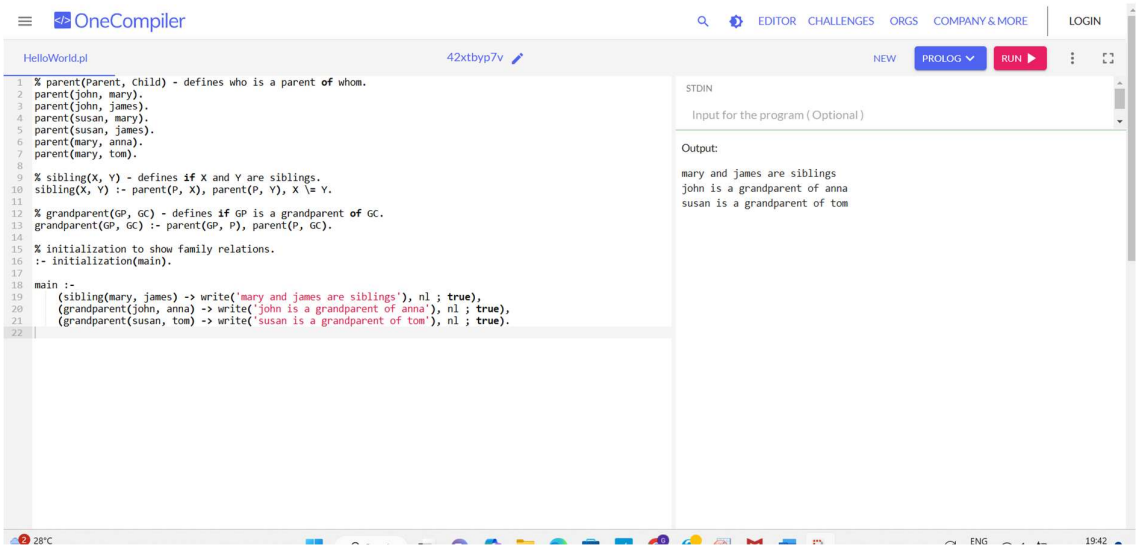

OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% teacher_student(Teacher, Student) - defines which teacher teaches which student.
teacher_student(john, alice).
teacher_student(john, bob).
teacher_student(mary, charlie).
teacher_student(mary, diana).
% Query to find all students of a teacher when loaded.
:- initialization(main).
main :-
teacher_student(john, Student),
write('Student = '), write(Student), nl, fail.
STDIN
Input for the program (Optional)
Output:
Student = alice
Student = bob
warning: /box/main.pro:8: user directive failed

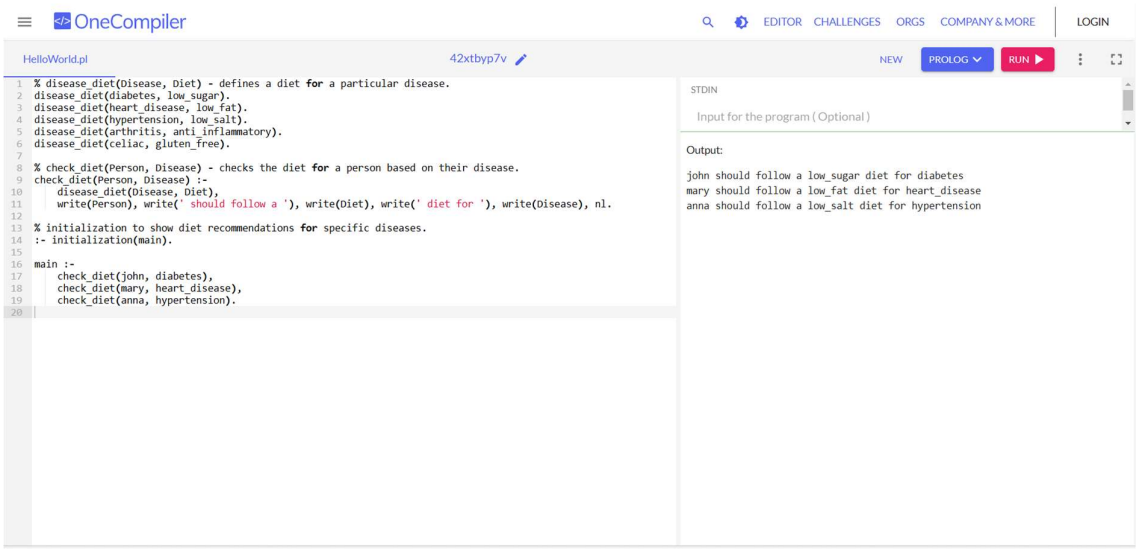

OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% fruit_color(Fruit, Color) - defines the color of each fruit.
fruit_color(apple, red).
fruit_color(banana, yellow).
fruit_color(grape, purple).
fruit_color(orange, orange).
% Query to find all fruits and their colors when loaded.
:- initialization(main).
main :-
fruit_color(Fruit, Color),
write(Fruit), write(' is '), write(Color), nl,
fail.
STDIN
Input for the program (Optional)
Output:
apple is red
banana is yellow
grape is purple
orange is orange
warning: /box/main.pro:8: user directive failed


OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% planet_type(Planet, Type) - defines the type of each planet.
planet_type(mercury, terrestrial).
planet_type(venus, terrestrial).
planet_type(earth, terrestrial).
planet_type(jupiter, gas_giant).
% Query to find all planets and their types when loaded.
:- initialization(main).
main :-
planet_type(Planet, Type),
write(Planet), write(' is a '), write(Type), write(' planet'), nl,
fail.
STDIN
Input for the program (Optional)
Output:
mercury is a terrestrial planet
venus is a terrestrial planet
earth is a terrestrial planet
jupiter is a gas_giant planet
warning: /box/main.pro:8: user directive failed


OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% hanoi(N, From, To, Aux) - solves the Towers of Hanoi for N disks.
hanoi(0, _, _, _) :- !.
hanoi(N, From, To, Aux) :-
N > 0, N1 is N - 1,
hanoi(N1, From, Aux, To),
write('Move disk from '), write(From), write(' to '), write(To), nl,
hanoi(N1, Aux, To, From).
% Initialization to display moves for 3 disks.
:- initialization(main).
main :-
hanoi(3, 'A', 'C', 'B').
STDIN
Input for the program (Optional)
Output:
Move disk from A to C
Move disk from A to B
Move disk from C to B
Move disk from A to C
Move disk from B to A
Move disk from B to C
Move disk from A to C


OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% bird_can_fly(Bird) - defines whether a bird can fly.
bird(canary).
bird(penguin).
bird(eagle).
can_fly(canary).
can_fly(eagle).
cannot_fly(penguin).
bird_can_fly(Bird) :-
can_fly(Bird),
write(Bird), write(' can fly'), nl.
bird_can_fly(Bird) :-
cannot_fly(Bird),
write(Bird), write(' cannot fly'), nl.
% Initialization to test with a particular bird.
:- initialization(main).
main :-
bird_can_fly(canary),
bird_can_fly(penguin),
bird_can_fly(eagle).
STDIN
Input for the program (Optional)
Output:
canary can fly
penguin cannot fly
eagle can fly


OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% parent(Parent, Child) - defines who is a parent of whom.
parent(john, mary).
parent(john, james).
parent(susan, mary).
parent(susan, james).
parent(mary, anna).
parent(mary, tom).
% sibling(X, Y) - defines if X and Y are siblings.
sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.
% grandparent(GP, GC) - defines if GP is a grandparent of GC.
grandparent(GP, GC) :- parent(GP, P), parent(P, GC).
% initialization to show family relations.
:- initialization(main).
main :-
(sibling(mary, james) -> write('mary and james are siblings'), nl ; true),
(grandparent(john, anna) -> write('john is a grandparent of anna'), nl ; true),
(grandparent(susan, tom) -> write('susan is a grandparent of tom'), nl ; true).
STDIN
Input for the program (Optional)
Output:
mary and james are siblings
john is a grandparent of anna
susan is a grandparent of tom


OneCompiler
EDITOR CHALLENGES ORGS COMPANY & MORE
LOGIN
HelloWorld.pl
42xtbyp7v
NEW
PROLOG
RUN
% disease_diet(Disease, Diet) - defines a diet for a particular disease.
disease_diet(diabetes, low_sugar).
disease_diet(heart_disease, low_fat).
disease_diet(hypertension, low_salt).
disease_diet(arthritis, anti_inflammatory).
disease_diet(celiac, gluten_free).
% check_diet(Person, Disease) - checks the diet for a person based on their disease.
check_diet(Person, Disease) :-
disease_diet(Disease, Diet),
write(Person), write(' should follow a '), write(Diet), write(' diet for '), write(Disease), nl.
% initialization to show diet recommendations for specific diseases.
:- initialization(main).
main :-
check_diet(john, diabetes),
check_diet(mary, heart_disease),
check_diet(anna, hypertension).
STDIN
Input for the program (Optional)
Output:
john should follow a low_sugar diet for diabetes
mary should follow a low_fat diet for heart_disease
anna should follow a low_salt diet for hypertension