Prolog Master Solutions

Complete worked answers in a compact phone layout

Source: PrologMasterSolutions.pl | Sections: 4 | Cards: 43
Generated: 2026-05-22 06:44

Structured Prolog solutions grouped by class and exercise.

Sections

Prolog 1

Database.

```
company(abc).
company(klm).
company(mno).
```

woman

```
woman(anna).
woman(maria).
woman(julia).
woman(ewa).
woman(joanna).
woman(lena).
woman(teresa).
```

man

```
man(jan).
man(karol).
man(piotr).
man(tomasz).
man(lukasz).
man(marek).
man(jozef).
```

employee(Name, Surname, Company, YearsOfWork).

```
employee(anna, klimczak, company(abc), 10).
employee(anna, maj, company(abc), 1).
employee(maria, jankowska, company(abc), 12).
employee(julia, klimczak, company(abc), 4).
employee(jan, kowal, company(abc), 21).
employee(karol, lis, company(abc), 5).
employee(anna, lis, company(klm), 12).
employee(piotr, bednarek, company(klm), 8).
employee(tomasz, bednarek, company(klm), 2).
employee(ewa, wilk, company(klm), 3).
employee(ewa, lipiec, company(klm), 7).
employee(lukasz, polak, company(klm), 11).
employee(marek, doba, company(klm), 8).
employee(anna, just, company(mno), 22).
employee(joanna, wilk, company(mno), 16).
employee(piotr, czekaj, company(mno), 4).
employee(maria, wilczak, company(mno), 16).
employee(piotr, kawa, company(mno), 14).
employee(marek, czubak, company(mno), 5).
employee(marek, lis, company(mno), 4).
```

Ex. 1a - ex1a

There is a woman whose name is lidia.

```
ex1a :- woman(lidia).
```

Ex. 1b - ex1b

There is an employee whose name is jan.

```
ex1b :- employee(jan, _, _, _).
```

Ex. 1c - ex1c

There is an employee whose name is jozef.

```
ex1c :- employee(jozef, _, _, _).
```

Ex. 1d - ex1d

There is an employee whose surname is karolak.

```
ex1d :- employee(_, karolak, _, _).
```

Ex. 1e - ex1e

There is an employee in company abc whose surname is maj.

```
ex1e :- employee(_, maj, company(abc), _).
```

Ex. 2a - anna_surname

Surnames of all employees whose name is anna.

```
anna_surname(Surname) :-
    employee(anna, Surname, _, _).
```

Ex. 2b - lis_name_years

Names and years of work of all employees whose surname is lis.

```
lis_name_years(Name, Years) :-
    employee(Name, lis, _, Years).
```

Ex. 2c - lis_company

Company names with employees whose surname is lis.

```
lis_company(CompanyName) :-
    employee(_, lis, company(CompanyName), _).
```

Ex. 2d - klm_employee

Names and surnames of all employees of company klm.

```
klm_employee(Name, Surname) :-
    employee(Name, Surname, company(klm), _).
```

Ex. 2e - man_employee

Names and surnames of all employees who are men.

```
man_employee(Name, Surname) :-
    employee(Name, Surname, _, _),
    man(Name).
```

Ex. 3a - employee_abc

True when a person named X Y is an employee of company abc.

```
employee_abc(X, Y) :-
    employee(X, Y, company(abc), _).
```

Ex. 3b - emplOther

True when a person named X Y is an employee of a company other than abc.

```
emplOther(X, Y) :-
    employee(X, Y, Company, _),
    Company \= company(abc).
```

Ex. 3c - emplWoman

True when a person named X Y is a woman employee.

```
emplWoman(X, Y) :-
    employee(X, Y, _, _),
    woman(X).
```

Ex. 3d - longEmpl

True when a person named X Y works at Z for at least 10 years.

```
longEmpl(X, Y, Z) :-
    employee(X, Y, Z, Years),
    Years >= 10.
```

Ex. 3e - bonus

True when employee X Y is entitled to bonus B.

```
bonus(X, Y, B) :-
    employee(X, Y, _, Years),
    B is Years * 150.
```

Prolog 2

Ex. 1 - max2

Z is the maximum of X and Y.

```
max2(X, Y, X) :-
    X >= Y.
max2(X, Y, Y) :-
    Y > X.
```

Ex. 2 - isList

True when X is a proper list.

```
isList([]).
isList([_|Tail]) :-
    isList(Tail).
```

Ex. 3 - isMember

True when X is an element of list L.

```
isMember(X, [X|_]).
isMember(X, [_|Tail]) :-
    isMember(X, Tail).
```

Ex. 4 - myLast

True when X is the last element of list L.

```
myLast(X, [X]).
myLast(X, [_|Tail]) :-
    myLast(X, Tail).
```

Ex. 5 - multiply

L contains products of corresponding elements of L1 and L2.

```
multiply([], [], []).
multiply([X|Xs], [Y|Ys], [Z|Zs]) :-
    Z is X * Y,
    multiply(Xs, Ys, Zs).
```

Ex. 6 - mySum

S is the sum of the numbers from list L.

```
mySum([], 0).
mySum([X|Xs], S) :-
    mySum(Xs, Rest),
    S is X + Rest.
```

Ex. 7 - myLength

N is the length of list L.

```
myLength([], 0).
myLength([_|Xs], N) :-
    myLength(Xs, Rest),
    N is Rest + 1.
```

Prolog 3

Ex. 1 - element_at

X is the K-th element of list L, using 1-based indexing.

```
element_at(X, [X|_], 1).
element_at(X, [_|Tail], K) :-
    K > 1,
    K1 is K - 1,
    element_at(X, Tail, K1).
```

Ex. 2 - dupli

L2 is obtained from L1 by duplicating all elements.

```
dupli([], []).
dupli([X|Xs], [X, X|Ys]) :-
    dupli(Xs, Ys).
```

Ex. 3 - my_reverse

L2 is obtained from L1 by reversing the element order.

```
my_reverse(L1, L2) :-
    my_reverse_acc(L1, [], L2).
```

my_reverse_acc

```
my_reverse_acc([], Acc, Acc).
my_reverse_acc([X|Xs], Acc, Result) :-
    my_reverse_acc(Xs, [X|Acc], Result).
```

Ex. 4 - end

L2 is obtained from L1 by adding X at the end.

```
end(X, [], [X]).
end(X, [Y|Ys], [Y|Result]) :-
    end(X, Ys, Result).
```

Ex. 5 - remove_at

X is the K-th element of L and R is L without that element.

```
remove_at(X, [X|Tail], 1, Tail).
remove_at(X, [Y|Tail], K, [Y|Rest]) :-
    K > 1,
    K1 is K - 1,
    remove_at(X, Tail, K1, Rest).
```

Ex. 6 - take

L2 contains the first N elements of L1.

```
take(_, 0, []).
take([X|Xs], N, [X|Ys]) :-
    N > 0,
    N1 is N - 1,
    take(Xs, N1, Ys).
```

Ex. 7 - my_flatten

L2 is obtained from L1 by recursively flattening nested lists.

```
my_flatten([], []).
my_flatten([X|Xs], Flat) :-
    is_list(X),
    !,
    my_flatten(X, FlatX),
    my_flatten(Xs, FlatXs),
    append(FlatX, FlatXs, Flat).
my_flatten([X|Xs], [X|FlatXs]) :-
    my_flatten(Xs, FlatXs).
```

Prolog 4

Ex. 1 - read_and_display

Reads a term ended with a dot and displays it.

```
read_and_display :-
    write('Enter text as a Prolog term, ended with a dot: '),
    read(Text),
    write(Text),
    nl.
```

Ex. 2 - mean_two_numbers

Calculates the arithmetic mean of two numbers.

```
mean_two_numbers :-
    write('First number: '),
    read(A),
    write('Second number: '),
    read(B),
    Mean is (A + B) / 2,
    write('Mean: '),
    write(Mean),
    nl.
```

Ex. 3 - circle_values

Calculates circumference and area for a given radius.

```
circle_values :-
    write('Radius: '),
    read(Radius),
    Circumference is 2 * pi * Radius,
    Area is pi * Radius * Radius,
    write('Circumference: '),
    write(Circumference),
    nl,
    write('Area: '),
    write(Area),
    nl.
```

Ex. 4 - read_items_until_stop

Reads items until the stop term is entered, then displays the list.

```
read_items_until_stop :-
    read_items_until(stop, Items),
    write(Items),
    nl.
```

read_items_until

```
read_items_until(Stop, Items) :-
    write('Item: '),
    read(Item),
    (   Item == Stop
    ->  Items = []
    ;   Items = [Item|Rest],
        read_items_until(Stop, Rest)
    ).
```

Ex. 5 - write_text_to_fixed_file

Reads one line from the user and saves it in a fixed file.

```
write_text_to_fixed_file :-
    FileName = 'prolog_user_text.txt',
    write('Text: '),
    read_line_to_string(user_input, Text),
    setup_call_cleanup(
        open(FileName, write, Stream),
        write(Stream, Text),
        close(Stream)
    ).
```

Ex. 6a - copy_uppercase_fixed

Copies a fixed input file to a fixed output file and uppercases all letters.

```
copy_uppercase_fixed :-
    uppercase_file('prolog_input.txt', 'prolog_output_upper.txt').
```

Ex. 6b - copy_uppercase_read

Reads input and output file names, then uppercases the copied content.

```
copy_uppercase_read :-
    write('Input file name, ended with a dot: '),
    read(InputFile),
    write('Output file name, ended with a dot: '),
    read(OutputFile),
    uppercase_file(InputFile, OutputFile).
```

uppercase_file

```
uppercase_file(InputFile, OutputFile) :-
    read_file_to_string(InputFile, Content, []),
    string_upper(Content, UpperContent),
    setup_call_cleanup(
        open(OutputFile, write, Stream),
        write(Stream, UpperContent),
        close(Stream)
    ).
```