UPPSALA UNIVERSITY
Department of Informatics and Media

Declarative Problem Solving Methods

Tasks for Laboratory Works and Lessons
Fall 2018

Content

Laboratory work 1

Preparations:
Read chapter 1.1-1.4, 2.1-2.2 in Bratko's book 2001 or 1.1-1.5, 2.1-2.2 in Bratko's book 2012 so that you understand the following: The syntax for Prolog programs, facts, rules, questions, atom, variable, instantiation of variables, matching of terms, satisfiable and unsatisfiable answers to questions, succeeding and failing goals.

Task 1
A number of programs are included in Prolog. These are called built-in programs or built-in predicates. There is one predicate for matching terms, =, and one for the evaluation of arithmetical expressions, `is`.

After ?- it is possible to ask questions. Test the following questions:

a) `john = john.`
b) `X = george.`
c) `X = Y, Y = maria.`
d) `date(X,Y,Z) = date(99,12,01).`
e) `date(X,8,Y) = date(2011,Z,W).`
f) `date(2010,10,X)= date(2011,10,5).`
g) `X is 5 + 4.`
h) `X is (5+4)*(5-3).`
i) `X is 5+4*5-3.`
j) `10 is 5*2.`
k) `12 is X*3.`
l) `write('Hello World').`
m) `write(Variable).`

It is possible to choose between getting one or several answers to a question. Try getting more answers for some of the questions above by using semicolon, i.e. **;**, after the answer.

Task 2
a) Write the following program that describes the relation minimum between three numbers, in a file using a text editor. The minimum of the first two arguments is the third element. Save the program using a proper name.

```
minimum(X,Y,X):-
     X=<Y.
minimum(X,Y,Y):-
     X>=Y.
```

b) Use Consult/Reconsult to load the program into the memory. After **?-** you pose the questions to Prolog. Give the following questions:

```
?- minimum(5,17,Min).
?- minimum(12,12,Min).
?- minimum(51,7,Min).
?- minimun(13,4,4).
?- minimun(13,4,13).
?- minimum(abc,abd,Min).
```

c) Change the program `minimum` so that @=< and @>= are used instead of =< and >= to compare terms. Consult the file again and test the program by using the questions in b) again.

d) Let Prolog give several answers by using semicolon (;) after the answer. Why does Prolog sometimes give one answer and sometimes two?

e) In SICStus Prolog it is possible to use the built-in predicate `trace` to be able to see Prolog's execution step by step. Write the question

```
?- trace, minimum(6,2,Z).
```

Use return to continue to follow the evaluation.

Comment: Except getting the answer to a question it is possible to follow the execution of the question. When you choose to run `trace` in Prolog the evaluation is presented stepwise. This is especially useful when debugging programs.

f) Change the program so it always gives only one answer for the questions in b). Use `trace` again.

Task 3
The following program was introduced at lecture 1 or 2. Test the program by asking which species are mammals according to the program. Continue by asking whether humans are mammals. Follow the execution with `trace`.

```
mammal(X):-
        viviparous(X), % give birth to living little ones
     carnivore(X). % meat-eater

viviparous(lion).
viviparous(tiger).

carnivore(Y):-
     has_sharp_teeth(Y),
     has_claws(Y).

has_sharp_teeth(lion).
has_sharp_teeth(tiger).

has_claws(tiger).
has_claws(lion).
```

Task 4
Execute the following program. Follow the evaluation with `trace` .

```
% part(X,Y) Y is a part of X
part(bike,saddle).
part(bike,wheel).
part(wheel,spoke). % in Swedish eker
part(wheel,tyre).
part(tyre,inner_tube).
part(inner_tube,valve_rubber).

% part_of_bike(X) X is a part of a bike or a part of a part of the bike
part_of_bike(X):-
        part(bike,X).
part_of_bike(X):-
        part(Y,X), part_of_bike(Y).
```

Task 5
a) Describe your family using facts for

```
father(X,Y)         % X is the father to Y
mother(X,Y)         % X is the mother to Y
man(X)              % X is a man
woman(X)            % X is a woman
```

b) Continue the description by stating rules for the following relations:

```
grandfather(X,Y)    % X is the grandfather to Y
                    % start with defining the father to a father
                    % continue with the father to a child
aunt(X,Y)           % X is the aunt to Y,
                    % i.e. a sister to your mother or father
uncle(X,Y)          % X is the uncle to Y
                    % i.e. a brother to your mother or father
parent(X,Y)         % X is the parent to Y
ancestor(X,Y)       % X is the ancestor to Y
```

c) Test the database by asking questions about your family. If some answers are not correct you should of course update the program.

d) Pose question about your family again, mainly to the predicates in b), and use `trace` so you can follow the evaluations stepwise. Continue until you really understand how the execution works.

Task 6
a) Put in more people from your family so you can test the program regarding cousin and second cousin. Definitions are given in the comments to the predicates.

b) Put the following clauses into your program:
```
% sibling(X, Y)
% X and Y are siblings if they have the same parent
% (includes half brothers and half sisters).
sibling(X,Y):-
     parent(Z,X),
     parent(Z,Y).

% cousin(X, Y)
% X and Y are cousins if X and Y:s parents are siblings.
cousin(X,Y):-
     parent(Z,X),
     parent(W,Y),
     sibling(Z,W).

% second_cousin(X, Y)
% X and Y are second cousins if X and Y:s parents are cousins.
second_cousin(X,Y):-
     parent(Z,X),
     parent(W,Y),
     cousin(Z,W).
```

c) Test the program and use `trace` to be able to follow the execution. Change the program if it doesn't give the correct answers.

Laboratory work 2

Preparations:

• Read chapter 2.3-2.6, 3 in Bratko 2001 and 2.3-2.5, 3 in Bratko 2012 so that you understand the following concepts: term, structure, functor, arity, unification of terms, lists, `[Head|Tail]`, and arithmetic operations.

• Try to solve the exercises before coming to the laboratory work.

In the tasks below you need to be able to compare and to calculate. There are built-in operators for numbers such as, e.g., >, <, >=, =<, and =\= (not equal). Moreover, there is a special predicate for evaluation of arithmetic expressions called `is`. As a goal or condition it is possible to write, e.g., `Z is X+Y`. When executing this predicate the variable `Z` is bound to the sum of `X` and `Y`. If all three arguments are bound to a value a comparison is done.

Note! Use the debugger, e.g. through `trace`, when you are testing your programs. In that way, it is easy to study how Prolog works and how your program functions.

Task 1

The built-in predicate `mod` gives the remainder of an integer division. For example, `X is 5 mod 2` means that 5 is divided with 2 and the remainder will be 1, i.e., `X=1`. The example `Y is 6 mod 3` gives `Y=0`.

Use `mod` to write
a) a program that concludes whether an integer is even
b) a program that concludes whether an integer X is evenly divisible with a given integer Y.

Task 2

To express that two arithmetic expressions have different values the operator =\= can be used. Example: `1+2 =\= 1+3` or `1 =\= 11 mod 3`.

Utilise `mod` and =\= to write
a) a program that concludes whether an integer is odd
b) a program that concludes whether an integer X is not evenly divisible with a given integer Y.

Task 3

Write a program that calculates whether a given year is a leap-year. The prerequisite for a leap-year is that the year is evenly divisible by 4 and not by 100. A special condition applies for those numbers evenly divisible with 100, since these also have to be divisible with 400 to be categorised as a leap-year. Thus, 1800 and 1900 are not leap-years, but 2000 is.

Task 4
Write a program that divides a list in two other lists, where the first one consists of the two first elements in the given list and the other one of the following elements.
Example: the list [25,42,13,7,10] is divided into the lists [25,42] and [13,7,10].

Task 5
Write a program that succeeds if the two words `fond` and `of` can be found in a list. If not both words are included the question should fail.

Task 6
Write a program that takes every second element in a list and puts them in a new list.
Example:

```
?- every_second([a,d,e,i,d,m,v],List)
List = [d,i,m]
```

Test the program with lists of different lengths.

Task 7
The aim with the game Hilo is to let a person guess a number, which value a computer (or a person) thinks of. For every guess of the player the answer from the computer should be:
To big!, Too small!, or Correct!.

Write a first Hilo version. Store the number the computer should "think of" in a predicate `secret_number(Y)` in the beginning of the program file.
Example:

```
secret_number(48).
```

The person that should guess gives the question `guess(X,Answer)` to Prolog. In the question a value for `X` is given, i.e. the guess.
Example:

```
?- guess(23,Answer).
```

Write the program `guess(X,Answer).`

Example:

```
?-guess(100,Svar).
Answer = Too big!

?-guess(50,Answer).
Answer = Too big!

?-guess(40,Answer).
Answer = Too small!

?-guess(45,Answer).
Answer = Too small!

?-guess(48,Answer).
Answer = Correct!
```

Task 8
Write a more advanced Hilo-program that gives the player a possibility to guess numbers without posing a question to Prolog for every guess. When the program has started with the call `guess_a_number` the user guesses until the correct answer is given. Use the same method to store the number that the computer is "thinking of" as in task 7.

In this program you should use `write(X), nl` (new line) to write the result and `read(Y)` to get the guess from the player. The in-built predicates for input and output are written as conditions in the program and these are evaluated as true.

Example: Assume we have a recursive predicate `p/2` that reads integers until a given decision is fulfilled. See the following as a pattern for such a program:

```
p(X,X):-
        write('Correct!!!'),
        nl.
p(X,Y):-
        ...,
        write('Give an integer: '),
        nl,
        read(Z),
        ...,
        p(X,Z).
```

In a Hilo-program with `read` and `write` the communication will be:

```
?- guess_a_number.
Give an integer: 100.
Too big!
Give an integer: 50.
Too big!
Give an integer: 40.
Too small!
Give an integer: 45.
Too small!
Give an integer: 48.
Correct!
```

Thus, the task is to write the recursive program `guess_a_number` that asks for new guesses until the correct one is given.

Laboratory work 3

Preparations:

• Read chapter 3, 6 and 7 in Bratko's book from 2001 or chapter 3 and 6 in Bratko 2012 so that you understand the following predicates: `read, write, tell, told, see, seen, consult, reconsult, assert, retract, retractall, bagof, setof,` and `findall`.

• Try to solve the exercises before coming to the laboratory work.

Task 1

Write a program `change_nth(N,Elem,List,Newlist)`, which is a program that changes the n:th element in `List` to `Elem`. The result is `Newlist`.

Example:

```
?-change_nth(3,4,[1,2,6,8,16], Newlist).
Newlist = [1,2,4,8,16]
```

Task 2

Assume that information about books is stored in the predicate
`book(Title, Author_List, Year_for_publishing, Company).`

The following database can be found in The Student portal .

```
%book(Title, Author_list, Year_for_publishing, Company)
book('Mathematical Logic',
    ['S.C. Kleene'],
    1967, 'John Wiley & sons').
book('Logic: Form and Function',
    ['J.A. Robinson'],
    1979, 'The University Press Edinburgh').
book('Lisp',
    ['P. Winston'],
    1981, 'Addison Wesley').
book('Lisp',
    ['P. Winston', 'B. Horn'],
    1989, 'Addison Wesley').
book('En match med Prolog',
    ['A-L. Johansson', 'A. Eriksson-Granskog', 'A. Edman'],
    1985, 'Studentlitteratur').
book('Applications of Prolog For You',
    ['J. Lucas'],
    2016,' CreateSpace Independent Publishing Platform').
book('Programming in Prolog. Using the ISO Standard',
    ['W.F. Clocksin, C.S. Mellish'],
    2003, 'Springer Verlag').
book('Prolog Versus You',
    ['A-L. Johansson', 'A. Eriksson-Granskog', 'A. Edman'],
    1989, 'Springer Verlag').
book('Logic Programming with Prolog',
    ['M. Bramer'],
    2013, 'Springer Science & Business Media').

book('Prolog Programming for Artificial Intelligence',
    ['I. Bratko'],
    2001, 'Addison Wesley').
```

```
book('Prolog Programming for Artificial Intelligence',
    ['I. Bratko'],
    2012, 'Pearson Education Limited').
book('The Art of Prolog',
    ['L. Sterling', 'E. Shapiro'],
    1994, 'The MIT Press').
book('Learn Prolog Now',
    ['P. Blackburn', 'J. Bos', 'K. Striegnitz'],
    2006, 'College Publications').
book('Artificial Intelligence',
    ['P. Winston'],
    1998, 'Addison Wesley').
book('Artificial Intelligence a Guide to Intelligent Systems',
    ['M. Negnevitsky'],
    2004, 'Addison Wesley').
book('A Modern Approach to Artificial Intelligence',
    ['S. Rusell', 'P. Norviq'],
    2016, 'Pearson International').
```

Define the predicate `find_book(Author, Title)`. The predicate `find_book` is used to find a title (or titles) on books that a person is author or co-author too.
Example:

```
?-find_book('A. Edman', Title).
Title='En match med Prolog';
Title='Prolog Versus You';
no
```

Task 3
Information about journals are stored in the predicate `journal(Name, Subject_list).`
Example:

```
journal('AI Magazine',['AI','expert systems', logic, 'logic
     programming','functional programming', robotics, learning, 'speach
     recognition', 'automatic deduction']).
journal('Formal methods',[logic,'program synthesis', 'program
     transformation','functional programming', 'logic
     programming','derivation editors']).
journal('Exert Systems',['knowledge acquisition', 'knowledge modelling',
     'knowledge representation', reasoning, 'reasoning with uncertainty',
     explanations, testing, 'learning support']).
```

These clauses can be found in a file in The Student portal. Write a program called `find_journal(Sub_list,Name)` which finds the journals that are related to the subjects asked for. These subjects are included in `Sub_list`. All elements in `Sub_list` should be in the list describing a journal's special subjects.

Example:

```
?-find_journal([learning,'expert systems','logic programming'], Journal).
Journal = AI Magazine
```

Task 4
Write a program that calculates the number of positive integers, negative integers, and zeros in a list of integers.

Example:
```
?- calculate([5,-4,-3,2,5,0,0,-3],Positive,Negative,Zero).
Positive=3, Negative=3, Zero=2
?- calculate([1,-1,-5],Positive,Negative,Zero).
Positive=1, Negative=2, Zero=0.
```

Task 5
Define a relation between a list and the mean-value of the numbers in the list.

Task 6
Given the clauses `book` in task 2 above, formulate `setof/bagof/findall` –questions to find the following:
a) Which books are written before 1980?
b) Which books are written during 1985 – 1995?
c) Which companies can be found in the given database?
d) Which titles are found in the database?
e) Which are the authors represented in the database?
f) Which titles have only one author?
g) Which titles have more than one author and who are in this case the authors?
h) Create a list where every element is a structure, `(Company,Title)`. All companies with their titles should be in the list and the list should be sorted - first after company and then after book title.

Task 7

It is possible to change predicates during the program execution. But to be able to change Prolog's database during the execution the predicates that you would like to change have to be defined as dynamic. In this case `journal` should be updated and therefore defined to be dynamic. This can be done by including

```
:- dynamic journal/2.
```

first in the program file. By this you specify to Prolog that the predicate `journal` could be updated and that the predicate has two arguments. The clause has no conclusion, only a condition, and then it is seen as a directive.

a) Add a journal during the execution using

```
assert(journal('Computational Linguistics',
     [ 'syntactic formalism', morphology, 'mathematical linguistics']))。
```

b) Delete the journal with the name `'Formal methods'`. This can be achieved using

```
retract(journal('Formal methods',Subjects)).
```

Use

```
?- listing(journal).
```

to check the current database.

c) Let the Prolog system store your journals in a file. Remember that if it should be possible to alter the relations they have to be defined as dynamic. Store this directive first in the file. Example:

```
     :
     tell('Journals.pl'), % open the file Journals.pl for output and
                          % direct the output to the file
     write(':- dynamic journal/2.')
                          % states that the predicate should be dynamic
     nl,                  % new line
     listing(journal),    % the relation journal is listed on the file
     told.                % closes the file
```

Laboratory work 4

Preparations:

• Read chapter 5 in the book written by Bratko (in both the third and fourth edition) and practise by doing the tasks in the chapter.
• Try to solve the exercises before coming to the laboratory work.

Task 1

Express a relation between a list of numbers and a list with all the odd numbers from the given list. Note that there should only be one list with odd numbers, after backtracking no new list should be generated.

Task 2

The following clauses are given (can be found in The Student portal):

```
feline(X):-
     cat(X).
feline(X):-
     big_cat(X).
feline(X):-
     cheetah(X).
cat(X):-
     domestic_cat(X).
cat(X):-
     wildcat(X).

big_cat(X):-
     panther(X).
big_cat(X):-
     tiger(X).
big_cat(X):-
     leopard(X).
cheetah(X):-
     hunting_leopard(X).
```

```
tiger('Shere-Khan').
tiger('Amur').

panther('Bagheera').

domestic_cat('Mjau').

hunting_leopard('Juba').

colour('Jarvis','Yellow').
colour('Leo','Yellow').
colour('Louis','Yellow').
colour('Blackie','Black').

speckled('Jarvis').
speckled('Leo').
```

Extend the given database with a definition for leopard:
a) An animal is a leopard if the colour is yellow and the animal is speckled
b) An animal is a leopard if the colour is black and the animal is not speckled

You can use `\+` in the program.
The negation as failure is implemented in Prolog in the following way:

```
\+(X):- X,!,fail.
\+(X).
```

c) Give the answers to the following questions:

```
i)   ?- \+leopard('Louis').   ii) ?- \+leopard('Blackie').
iii) ?- \+speckled(Animal).   iv) ?- \+colour(Animal,'Brown').
v)   ?- \+colour(Animal,'Black').
```

d) Study the answers that Prolog generates. Explain why Prolog gives these answers. Are there questions that it is possible to expect another answers to in c)?

Task 3
Define a relation between two lists where one list should contain all the elements in the first list that starts with a given combination of letters. Use `name` and negation to solve the task.
Example:

```
?-beginning([bc,abc,a,abde,adb,abf],ab,New_list).
New_list=[abc,abde,abf].
```

Task 4
Write a program `delete(Elem,List1,List2)` which takes away all elements `Elem` from a list and gives a list without the given elements. Define the program according the following three alternatives:
a) a program without cut, i.e. !
b) a program with only green cuts
c) change the green cuts so they become red. Note that this influences the declarative interpretation of the program!

Task 5
Write a definition for a relation that is true if two terms are not identical. The operator '\==' should not be used in the program.

Task 6
Write a program `flatten(+List_of_lists,-List_of_elements)` that given a list with several levels creates a flat list. Write two program, one with cut and one without it.
Example:

```
?-flatten([a,b,[c,d],e,[f,g,[h,i,j],k]],L)
L=[a,b,c,d,e,f,g,h,i,j,k]
```

Laboratory work 5

F# Lab First Session

Task 1

Create an F# Application project, write the necessary code to print your name on the screen. Run using F5. Don't forget to end your code with a Console.ReadKey() so the console will stay on the screen and won't disappear before you see what was printed in the console.

```
open System
printfn, "%s"
Console.ReadKey() |>ignore
```

Task 2

Write the code for printing numbers 1 through 10 on the screen, on the same line, separated by spaces.

`Keywords: for, do, printf`

Task 3

Write a function to print a string on console. It should take a string argument and return nothing. You may use plain function arguments, or you may use currying. Call the function to print some text on the console.

`Keywords: let, printfn`

Task 4

Write a function 'printArr' that will print every value in the given array in separate lines on console. Test the function to print numbers 1 through 10 on the console.

`Keywords: let, for, in, do, seq`

Task 5*

Write a recursive function to calculate n! (factorial). You do not need to pay attention to negative input, but explicitly make sure the input argument is an integer. Use this function to print 10! on the console.

`Keywords: let, rec, int, match, with`

*Assignment 5: Solve this task individually and upload the solution to the student portal.

Task 6

Write a function ‘mymap’ that takes a unary function (function with 1 argument) and a sequence as arguments, and returns a new sequence where values are results of the given function for the values in the given **sequence**. Test this ‘mymap’ function by passing it a squaring (x*x) anonymous function and a sequence of numbers 5 through 15. Use the ‘printArr’ function you implemented at step 4 to display the results.

`Keywords: let, for, in, do, seq, fun`

Task 7

Write a function that takes an integer array, and calculates its standard deviation.

`Keywords: let, int, array, Seq.sum, Seq.length, Seq.map, fun, sqrt, float`

Laboratory work 6

F# Lab Second Session

Task 1

Write a function that takes a sequence and returns the sorted sequence. It should use the merge sort algorithm to do the sorting. Note that you may need to write two sub-functions (or helper functions) to implement the task in a declarative style (a split, a merge and the mergeSort function).

Here's the pseudocode of Merge Sort from Wikipedia[1]. Conceptually, a merge sort works as follows:

1. If the list is of length 0 or 1, then it is already sorted. Otherwise:
2. Divide the unsorted list into two sublists of about half the size.
3. Sort each sublist recursively by re-applying the merge sort.
4. Merge the two sublists back into one sorted list.

Note that this recursive algorithm will eventually return a sorted list.

`Keywords: let rec, match, Seq.length, Seq.take, Seq.skip, Seq.toList, Seq.append`

[1] https://en.wikipedia.org/wiki/Merge_sort, Accessed 2017-08-25

Lesson 1

Literature: Prolog Programming for Artificial Intelligence chapter 1.1-1.4, 2.1-2.2 in Bratko 2001, 1.1-1.5 , 2.1-2.2 in Bratko 2012.

Task 1

The following pairs of predicates should be matched, i.e. unified. If a matching succeeds the instantiations should be presented. If the matching does not succeed you should explain the reason to this.

a)	`p(a,X)`	and	`p(Y,b)`	b)	`p(a,X)`	and	`p(Y,Y)`
c)	`p(a,X,X)`	and	`p(Y,Y,b)`	d)	`q(a,X)`	and	`q(Y,f(Z))`
e)	`q(a,X)`	and	`q(Y,f(Y))`	f)	`q(X,Y,Z)`	and	`q(W,f(W, U))`
g)	`q(Y,Y)`	and	`q(Z,f(f(Z)))`				

Task 2

The following Prolog clauses are given for the following relations:

```
%father(X,Y)          X is the father to Y
%mother(X,Y)          X is the mother to Y
%man(X)               X is a man
%woman(X)             X is a woman
%married(X,Y)         X and Y are married
%parent(X,Y)          X is the parent to Y
%different(X,Y)       X and Y are different individuals

father(ingemar,anneli).
father(anders,katja).
father(ingemar,inge).
father(ingemar,owe).
father(inge,sara).
father(owe,jonathan).

mother(annmarie,anneli).
mother(annmarie,inge).
mother(annmarie,owe).
mother(anneli,katja).
mother(katja,linus).

man(ingemar).
man(anders).
man(inge).
man(owe).
man(linus).
man(jonathan).

woman(annmarie).
woman(anneli).
woman(katja).
woman(sara).


married(annmarie,ingemar).
married(bjorn,katja).
married(anneli,anders).
```

```
parent(X,Y):-
        father(X,Y).
parent(X,Y):-
        mother(X,Y).

different(X,Y):-
        X\==Y.
```

Write the following programs:

a) `son(X)              % X is a son`
b) `sister(X,Y)         % X is the sister to Y`
c) `mother_in_law(X,Y) % X is mother in law to Y`

d) Update the given definition of nephew so Prolog always gives correct answers when using it, independent of how the question is posed.

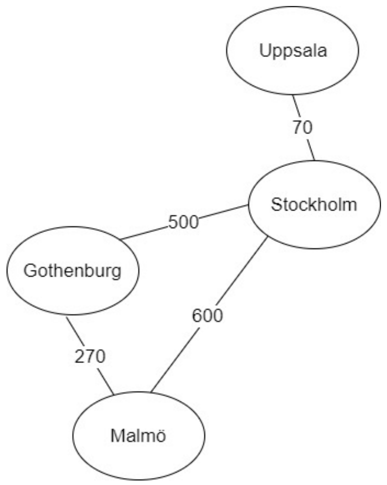
```
% nephew(X,Y)         X is the nephew to Y
nephew(X,Y):-
        parent(Z,X),
        parent(Z,W),
        father(W,Y).
```

Task 3
Write a Prolog program that describes which number is the smallest and which is the biggest of two given numbers. The relation can be called `minmax(X,Y,Min,Max)` where `Min` is the smallest of `X` and `Y` and `Max` is the biggest of the two.

To compare the differences, such as less than, less than or equal to, bigger than, and bigger than or equal, the following operators can be used: < , =< , > , >= .

Task 4
The following Prolog clauses are given for the relation `way/3`.

```
%way(City1, City2, Distance).
way('Stockholm', 'Uppsala', 70).
way('Stockholm', 'Malmö', 600).
way('Gothenburg', 'Stockholm', 500).
way('Gothenburg', 'Malmö', 270).
```



a) What are the answers to the following questions?

```
?- way('Stockholm', 'Uppsala', X).
?- way('Stockholm', 'Uppsala', 70).
?- way('Uppsala', 'Stockholm', 70).
?- way(X, Y, Z).
?- way(X, X, Z).
```

b) Write a program that checks whether there is a route between two cities:

```
%route(City1, City2).

?- route('Gothenburg', 'Stockholm').
yes
```

```
?- route('Gothenburg', 'Uppsala').
yes
```

Task 5
The following Prolog clauses are given for the relation `book/4`. A file with the given database can be found in The Student portal.

```
%book(Title, Author, Year_for_publishing, Company)
book('Mathematical Logic','S.C. Kleene',1967,'John Wiley & sons').
book('Logic: Form and Function','J.A. Robinson',1979,
     'The University Press Edinburgh').
book('Lisp','P. Winston',1981,'Addison Wesley').
book('Lisp','P. Winston, B.Horn',1989,'Addison Wesley').
book('En match med Prolog','A-L. Johansson, A.Eriksson-Granskog, A.Edman',
     1985,'Studentlitteratur').
book('Applications of Prolog For You','J. Lucas',2016,
     ' CreateSpace Independent Publishing Platform').
book('Programming in Prolog. Using the ISO Standard',
     'W.F. Clocksin, C.S. Mellish',2003, 'Springer Verlag').
book('Prolog Versus You', 'A-L. Johansson, A.Eriksson-Granskog, A.Edman',
     1989,'Springer Verlag').
book('Logic Programming with Prolog', 'M. Bramer', 2013,
     'Springer Science & Business Media').
book('Prolog Programming for Artificial Intelligence','I. Bratko',
     2001,'Addison Wesley').
book('Prolog Programming for Artificial Intelligence','I. Bratko',
     2012,'Pearson Education Limited').
book('The Art of Prolog','L. Sterling, E. Shapiro',1994,'The MIT Press').
book('Learn Prolog Now', 'P.Blackburn, J.Bos, K.Striegnitz',2006,
     'College Publications').
% See free online version at http://www.learnprolognow.org/
book('Artificial Intelligence','P. Winston',1998,'Addison Wesley').
book('Artificial Intelligence – a Guide to Intelligent Systems',
     'M. Negnevitsky',2011, 'Pearson Education Limited').
book('A Modern Approach to Artificial Intelligence', 'S. Rusell,
P.Norviq',2016, 'Pearson International').
```

a) For comparing terms the following built-in operators can be used:
X<Y describes that X is lesser than Y
X>=Y describes that X is greater or equal to Y
X\==Y expresses that X is a term different from Y.

Formulate questions so you can get the following answers
i) books printed before 1990
ii) books printed 2000 or later
iii) books printed later than 1990 and published by the book company Addison Wesley
iv) books written by I. Bratko
v) authors that have written at least two books

b) Which are Prolog's answers to the questions in i) to v) according to the `book` clauses above?

Lesson 2

Literature: Bratko, both third and fourth edition, chapter 2 and 3, including the exercises.

Task 1

Write a Prolog program that can find out whether three given lengths together can form a triangle. The sides, i.e. the lengths, can form a triangle if they are positive numbers and the sum of two of the sides always is bigger than the third.
Example:

```
?-triangle(2,3,4).
yes
?-triangle(1,2,3).
no
```

Task 2

Utilise the program in task 1 to decide whether a triangle is ordinary, isosceles (two sides have the same length), equilateral (all sides have the same length), or right-angled. A triangle is right-angled if the sum of the squares of two of the sides is equal as the square of the third side.

The relation can be expressed as `triangle(X,Y,Z,Type)` where `X,Y,Z` are the sides and `Type` is a word denoting the type. Example:

```
?-triangle(2,3,3,Type).
Type= isosceles
?-triangle(3,3,3,Type).
Type= equilateral
?-triangle(5,4,3,Type).
Type= right-angled
?-triangle(2,3,4,Type).
Type= ordinary
?-triangle(2,4,6,Type).
No
```

alternatively

```
?-triangle(2,4,6,Type).
Type = no triangle
```

Task 3
Write a program that investigates whether a number is evenly divisible with the prime factors 2, 3 or 5. The program should generate an answer describing if the number is divisible by all three, with two or with one of 2, 3 and 5. Write the program according a), b) or c).

a) A variable in the question is instantiated to a describing text. Example
```
?- divisible(30,Answer).
Answer = 'The number is evenly divisible with 2,3 and 5'
```

b) Three variables are instantiated to 2, 3 and/or 5 if these divide the given number. Example:
```
?- divisible(30, Answer1, Answer2, Answer3).
Answer1 = 2, Answer2 = 3, Answer3 = 5
```

c) A predicate, `evenly_divisible(X),` performs tests whether `X` is divisible with the three given prime numbers and the result is written utilising `write(X), nl` (new line). Example:
```
?- evenly_divisible(30).
30 is divisible with 2  3  5
```

Lists in Prolog

An empty list	`[]`
A non-empty list	`[Head|Tail]`
A list with 5 elements	`[X,Y,Z,U,V]`

<u>Patterns for list programs</u>
1. An empty list is the base case or halting case:
```
p([]).
p([Head|Tail]):-
        p(Tail).
```

2. A base case that is not an empty list:
```
p(X):-
        condition(X).
p([Head|Tail]):-
        p(Tail).
```

3. A list connected to calculation:
```
p(0,X).
p(N,[Head|Tail]):-
        N1 is N-1,
        p(N1,Tail).
```

An example program of type 3:
```
put_after_nth(0,X,L,[X|L]).
put_after_nth(N,X,[Head|Tail],[Head|T]):-
        N1 is N-1,
        put_after_nth(N1,X,Tail,T).

?-put_after_nth(2,yellow,[blue,red,green,white],List).
List= [blue,red,yellow,green,white]
```

Task 4

Write a recursive program defining the relation called `less_than_all(Element,List)`. The relation is satisfied when the element is less than all the elements in the list. Assume that the list elements are numbers.

Example:

```
?- less_than_all(3,[7,6,11,4]).
yes
```

Task 5

Write a recursive program that finds the fifth element in a list.

Example:

```
?- find_fifth([12,3,4,6,2,1,9,17],X).
X=2
```

Rewrite the program so it can find an arbitrary element in the list.

Example:

```
?- find(5,[12,3,4,6,2,1,9,17],X).
X=2
```

Lesson 3

Literature: Bratko 2001 chapter 3, 6, 7 and Bratko 2012 chapter 3, 6.

Task 1

Write a program `find_the_following(E,List,Follower)` which finds the element following the element `E` in `List`.

Example:

```
?- find_the_following(2,[12,3,4,6,2,9,1,17],Follower).
Follower=9
```

Task 2

Define `num_of_one_element(Element,List,Number)`, i.e. a relation between a list and the number of occurrences of the element `Element` in `List`.

Example:

```
?- num_of_one_element(4,[12,3,4,6,4,4],Number).
Number=3
```

Task 3

Write the program `delete_all(Element,List,Newlist), i.e.` a program that takes away all occurrences of `Element` from `List` constituting a `Newlist`.

Example:

```
?-delete_all(4,[12,3,4,6,2,4,9,1,4,17], Newlist).
Newlist=[12,3,6,2,9,1,17]
```

Task 4

Define the relation `delete_nth(N,List,Newlist),` where `Newlist` has the same elements as `List` except for the element number n which is removed. Example:

```
?- delete_nth(4, [12, 3, 4, 6, 2, 9, 1, 17], X).
X=[12, 3, 4, 2, 9, 1, 17]
```

Task 5

Write a program that reads two numbers, using the built-in predicate `read(X)`, and then calculate the mean value. The result should be presented using `write(Y)`. The built-in predicate `nl` gives a new line.

Example:

```
Give an integer: 23.
Give an integer: 37.
The mean value is:
30.
```

Task 6
Write a program that sum up a number of integers. Utilise the built-in predicates `read`, `write` and `nl` for the interaction. When the user has started the program he/she is asked for new numbers until the word stop is given.
Example:

```
Give an integer: 23.
Give an integer: 54.
Give an integer: 100.
Give an integer: stop.
Sum: 177
```

Task 7
Take the program from task 6 and develop it further by storing the input integers in a list. Then sum all the numbers in the list after the word stop has been read.

Lesson 4

Literature: Bratko 2001 chapter 6 and 7 and Bratko 2012 chapter 6.

Task 1

Given the following relative database, utilise `setof` to collect all the names of a) all fathers b) all sons c) all daughters d) all uncles e) all sons to a specified person.

```
%father(X,Y)          X is the father to Y
%mother(X,Y)          X is the mother to Y
%man(X)               X is a man
%woman(X)             X is a woman
%parent(X,Y)          X is the parent to Y
%brother(X,Y)         X is the brother to Y
%uncle(X,Y)           X is the uncle to Y

father(ingemar,anneli).
father(anders,katja).
father(ingemar,inge).
father(ingemar,owe).
father(inge,sara).
father(owe,jonathan).

mother(annmarie,anneli).
mother(annmarie,inge).
mother(annmarie,owe).
mother(anneli,katja).
mother(katja,linus).

man(ingemar).
man(anders).
man(inge).
man(owe).
man(linus).
man(jonathan).

woman(annmarie).
woman(anneli).
woman(katja).
woman(sara).

parent(X,Y):- father(X,Y).
parent(X,Y):- mother(X,Y).

brother(X,Y):-
        parent(P,X),
        parent(P,Y),
        man(X),
        X\==Y.

uncle(X,Y):-
        brother(X,Z),
        parent(Z,Y).
```

Task 2
Write a program that reads and then stores names and addresses on a file until the user writes the word stop.

Example:
```
Give a name: 'Anders Eriksson'.
Give an address: 'S:t Olofsgatan 5, 754 31 Uppsala'.
Give a name: 'Berit Ekbom'.
Give an address: 'Drottninggatan 7, 754 01 Uppsala'.
Give a name: stop.
The addresses are stored in the file Addresses.pl
```

When a name and an address is read the information should be saved in a relation `address(Name,Address)`. The built-in predicate `assert` can be used to store a relation in Prolog's database. Example:
```
p:-...,assert(address(X,Y)),....
```

When the user has written `stop` should all the relations `address(X,Y)` be stored in a file. If these relations should be able to be updated during a program execution these have to be defined dynamic. Store this information first in the file. Saving information in a file is done through
```
        :
        tell('Addresses.pl'),
        write(':-dynamic address/2.'),nl,
        listing(address),told.
```

Task 3
Write a program that can update a name and/or an address for a person and store it in the database.
Example:
```
Give the name and address that should be updated.
Give the name: 'Berit Ekbom'
Give the address: 'Drottninggatan 7, 754 01 Uppsala'
Give the new name: 'Berit Ekbom Olsson'
Give the new address: 'Skolstigen 5, 754 31 Uppsala'
The change is done. The addresses are stored in the file Addresses.pl
```

- Start with deleting all the relations `address` from the Prolog database.
- Use in the program `consult('Addresses.pl')` to get all the relations from the file Addresses.pl. into your Prolog database.
- Read data according the example above.
- Delete the relation that should be changed.
- Assert the new one.
- Save the relations, in the same way as in task 2, in the file.

Task 4
Write a program that reads a character and write the next character according to the ascii table.
Example:
```
Give a character: A
Answer: B
Give a character: 7
Answer: 8
```

Task 5
Write a program `plus` that uses the operator +, but before + is executed the program should test whether the user has given a correct value. If this is not the case a message should be given. If a correct value is given the program should deal with the problem in an appropriate way
Example:
```
?-plus(4,5,X).
X=9
?-plus(X,5,10).
X=5
?-plus(X,5,Y).
Cannot solve the problem.
```

Task 6
Write a program that reads a fact for father and then writes the information regarding the relation in "natural language". The program should use the given information and decide as little as possible in beforehand.
Example:
```
Give a fact for father: father('Karl Gustav', 'Victoria').
father to Victoria is Karl Gustav
```

You could utilise the built-in predicate =.. (example: `p(X,Y)=..[p,X,Y]`) or the predicates `functor` and `arg`.

Task 7
Write a program that reads a word and then creates a new one where all letters "a" are taken away. Use the built-in predicate `name`. Example:
```
?-word(abrakadabra,Word).
Word=brkdbr
```

Lesson 5

Literature: Bratko 2001 chapter 1-8 Bratko 2012 chapter 1-6 and 8.

Parts from the written exam 2008-11-08 and 2014-10-10

Task 1 Theory (6 + 4 points)

a) The derivation in Prolog is based on the resolution rule. Illustrate how the resolution rule works through deriving that literature('The girl with the dragon tattoo', fiction) is true when the following Horn clauses are given:

literature(X, fiction) <- check_fiction(X)

check_fiction(X) <- crime_novel(X), action(X, exciting)
check_fiction(X) <- novel(X), action(X, romantic)
check_fiction(X) <- novel(X), action(X, tragic)

crime_novel('The girl with the dragon tattoo') <-
action('The girl with the dragon tattoo', exciting) <-

b) When applying the resolution rule the terms *resolvent* and *parents to the resolvent* can be used. Explain these two terms and illustrate them utilising the derivation in **a)**.

Task 2 Unification (8 points)

Check whether the predicate pairs can be unified. If the unification succeeds show the instantiations, i.e. the variable values. If the unification is unsuccessful explain the reason to this.

a) `a(X,p,Y) = a(Z,Z,5).`

b) `b(f(f(X)),X,5) = b(Z,1,Y).`

c) `c(p(p(p(X))),p(Z)) = c(Z,X).`

d) `d(structure(X,Y),X,2)=d(Z,1,Y).`

e) `e(func(1,2),X,Y) = e((X,Y),1,2).`

f) `f([X,Y,Z|L1],[Y,Z]) = f([a,b,c],L2).`

g) `g([X,Y|X]) = g([2,4,2]).`

h) `h([X,Y|L]) = h([2,4,[2]]).`

Task 3 Rewrite programs (6 + 7 points)

a) Write a program `add_character(Char,Inlist,Outlist)` where `Outlist` consists of the same elements as `Inlist` but between every element the character `Char` has been included. Update the given program so it follows the given definitions.

Ex:
```
?- add_character(2,[1,3],Outlist).
Outlist = [1,2,3]

?- add_character(2,[1,3,5],Outlist).
Outlist = [1,2,3,2,5]

?- add_character(2,[1],Outlist).
no
```

```
add_character(Char,[],[]).
add_character(Char,[A,B| Rest],[A,Char,B| NewRest):-
      add_character(Char,Rest,NewRest).
```

b) The recursive program `position(Pos,Elem,List)` searches for the element `Elem` in `List` that has the position `Pos`. Update the program so it follows the given definition.

```
?- position(3,Elem,[12,3,5,8,9]).
Elem = 5

position(0,E,[Elem|_]).
position(Pos,Elem,[E|Rest]):-
      position(P,E,Rest),
      P is Pos – 1.
```

Task 4 Check the search space (6 + 6 points)

The predicate `elimininate(Elem,List,NewList)` defines the relation between `List` and `NewList` where all elements in `List` can be found in `NewList` except those elements that are equal to `Elem` .

```
eliminate(X,[],[]).
eliminate(X,[X|Rest],NewRest):-
      eliminate(X,Rest,NewRest).
eliminate(X,[Y|Rest],[Y|NewRest]):-
      eliminate(X,Rest,NewRest).
```

a) Give all alternative answers that the program can give on the following question:
```
?- eliminate(a,[a,b,a],List).
```

b) Rewrite the program in two different ways so that only correct answers are generated when asking Prolog for alternative answers.

Task 5 Work with a database – find all answers (3 + 3 + 4 + 5 points)

The given database describes some of the sights recommended for sightseeing in Uppsala.

```
% to_see(Name,Type,place(City,Position)).
to_see('The cathedral',church,place('Uppsala', 'city centre')).
to_see('Old Uppsala church',church,place('Uppsala','Gamla Uppsala')).
to_see('Old Uppsala hills','open-air museum',place('Uppsala','Gamla
   Uppsala')).
to_see('Old Uppsala museum',museum,place('Uppsala','Gamla Uppsala')).
to_see('Gustavianum',museum, place('Uppsala', 'city centre')).
to_see('Carolina Rediviva',museum, place('Uppsala', 'city centre')).
to_see('Uppsala university building',university,place('Uppsala','city
   centre')).
to_see('Upplandsmuseet',museum,place('Uppsala', 'city centre')).

% founded(Name,Year)
founded('The cathedral',1435).
founded('Old Uppsala church',1164).
founded('Old Uppsala museum',2000).
founded('Old Uppsala hills',500).
founded('Gustavianum',1620).
founded('Carolina Rediviva',1841).
founded('Upplandsmuseet',1760).
founded('Uppsala university building',1887).

%contents(Name,List_of_contents)
contents('The cathedral',['Reliquary casket of St Erik',
   'Gustav Vasas tomb', 'The treasury','The Sture costumes',
   'The golden gown of Queen Margareta of Denmark, Norway and Sweden']).
contents('Old Uppsala church',['altar-stone', 'crucifix',
   'baptismal font']).
contents('Old Uppsala museum',['Myth, Might and Man exhibition']).
contents('Old Uppsala hills',['Royal mounds', 'Tingshögen']).
contents('Gustavianum',['Augsburg art cabinet', 'Anatomical theatre',
   'Numesmatic cabinet', 'Egyptologic collection']).
contents('Carolina Rediviva',['Carta Marina', 'Silver Bible',
   'Muscial notation by Mozart', 'Emperers bible']).
contents('Upplandsmuseet',['Collections',' Walmstedts house', 'Disa
farm']).
contents('Uppsala university building',['assembly-hall', 'light dome',
   'University Senate room']).
```

Formulate questions, utilising `setof`, `bagof` or `findall`, to get the answers to the questions in **a) - d)**. You don't have to give the answers to the questions.

a) Which buildings can you visit in the city centre of Uppsala?
b) Which are the churches in Uppsala?
c) Which buildings are founded after 1500 in the database?
d) Which museums are founded before 1900 in the database? The museums should be sorted alphabetically in a list.

Task 6 Work with a database – writing programs (6 + 7 + 7 + 3 points)

Utilise the database in task 4!

a) Write a program that can answer in which building a special object can be found and where this building is situated.

Ex:
```
?- where_to_find('Silver Bible',Building,Where).
Building = Carolina Rediviva
Where = place(Uppsala,'city center')
```

b) Write a program that can present a special site according to the giving format. The program should use the built-in predicate write for the presentation of the information.

Ex:
```
?- present_info('Old Uppsala hills').

Old Uppsala hills is a open-air museum, founded 500, situated at Old
Uppsala in Uppsala.
Here you can see
  Royal mounds
  Tingshögen
```

c) It should be possible to update information regarding what you can see in a building. Write a program that solves this. The new information should be given in a list. Let these new objects start the List_of_contents in the database. This new information should be stored in the database and the old one should be deleted.

```
Ex:
?- update_obj_list('Gustavianum',['Physical cabinet', 'University
history
   exibition', 'Viking exibition']),listing(contents).

:
contents('Gustavianum', ['Physical cabinet', 'University history
exibition','Viking exibition', 'Augsburg art cabinet', 'Anatomical
theatre', 'Numesmatic cabinet', 'Egyptologic collection']).
```

d) To be able to execute the program update_obj_list in c) we must give a command to the Prolog interpreter. Which is the command, i.e. what should be included in the program and where should it be put in the program? Why must this command be included in the program?

Task 7 Trace a Prolog program (6 + 3 + 5 + 5 points)

```
program([],0).
program([E|Rest],X):-
      integer(E),
      program(Rest,X1),
      X is X1 + E.
program([E|Rest],X):-
      \+ integer(E),
      program(Rest,X).
```

a) If we ask the program the following, which value does the variable Result get?

```
      ?- program([abc,15,3,*,def,7,8,*],Result).
```

b) Explain in words what the program performs.

c) This program is not tail recursive. Why not? What's the point of having tail recursive programs?

d) Some programs that are not tail recursive can be rewritten so it will be tail recursive. Describe how you can rewrite such a program.

Appendix with built-in predicates in Prolog

`<`	less than
`=<`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`=`	the two arguments can be unified
`=/=`	the two arguments cannot be unified
`X is Y`	`X` gets the value that the arithmetical expression `Y` has, `Y` must contain instantiated variables
`= =`	the two arguments are identical
`\==`	the two arguments are not identical
`X+Y`	addition
`X-Y`	subtraction
`X*Y`	multiplication
`X/Y`	division
`X//Y`	integer division
`X mod Y`	gives the rest from the division between `X` and `Y`
`integer(X)`	is true if `X` is an integer
`atom(X)`	is true if `X` is an atom
`atomic(X)`	is true if `X` is a number or an atom
`read(X)`	reads a term
`write(X)`	writes a term
`nl`	gives a new line
`see(File)`	opens a file for reading or switches the input stream to `File`
`seen`	closes file
`tell(File)`	opens file for writing or switches the output stream to `File`
`told`	closes file
`listing(P)`	lists the predicate `P`
`assert(P)`	stores a predicate `P`
`retract(P)`	deletes a stored predicate `P`
`name(X,AsciiList)`	the relation between the term `X` and a list of Ascii-codes for the term.

Ex: `?-name(abö,L) ger L=[97,98,124]`
`?-name(X,[97,46,99,33,63,44]) gives X=a.c!?,`

`bagof(X,P(X),L)`

The first argument decides the elements in the output list, the second is the question, and the third is the list with the answers

`findall(X,P(X),L)`

The predicate works like bagof except that all variables in P are existentially quantified

`setof(X,P(X),L)`

The predicate works like bagof with the addition that the elements in L are sorted and duplicate items are deleted.

Lesson 6

Literature: Bratko 2001 chapter 1-8 and in Bratko 2012 chapter 1-6 and 8.

Written exam 2008-09-30

Task 1 Problem solving (6 points)

During the course we have mainly used the problem solving methods “divide-and-conquer” and “generate-and-test”. Describe these two methods. Discuss the advantages and disadvantages you can see when utilising them in a declarative programming language such as Prolog.

Task 2 Unification (8 points)

Check whether the predicate pairs can be unified. If the unification succeeds show the instantiations, i.e. the variable values. If the unification is unsuccessful explain the reason to this. **a)**

a) `info(X,a,Z,b) = info(Y,Z,Y,W).`

b) `data(f(X),Y,f(Z)) = data(Y,a,f(5)).`

c) `data(X,p(X),Z,4) = data(d(8,10),Y,W,W).`

d) `data(Y,p(X),a) = data(p(X),p(Y),Z).`

e) `info(X,Y,f(Z)) = info(5,f(X),Y).`

f) `lists(L,[1,2,3],N) = lists([a,b],[E,F|G],[X|L]).`

g) `lists([a,b],p([1,2],3)) = lists(L,p(X,Y,Z)).`

h) `lists(L,[1,2,3],N) = lists([a,b],[E|[G,H]],[L|[]]).`

Task 3 Rewrite programs (8 + 8 points)

Update the given programs so they follow the given definitions.

a) Write a program `add_elements(L1,L2,L3)` that takes two lists, `L1` and `L2,` and generates a new list with numbers, `L3`. The new list's elements should be the sum of the two elements in the same position in the given lists `L1` and `L2`. One list may have more elements than the other. In that case the end elements in the longer list should be included in `L3`.

Ex:
```
?- add_elements([1,2,3],[4,5],L3).
L3 = [5,7,3]

?- add_elements([1,2,3],[4,5,6,7,8],L3).
L3 = [5,7,9,7,8]
```

```
add_elements([],[],[]).
add_elements([],[E|R],R3):-
      add_elements([],R,R3).
add_elements([E1|R1],[E2|R2],R3):-
      E3 is E1 + E2,
      add_elements(R1,R2,[E3|R3]).
```

b) The recursive program `store_odd_numbers(L1,L2)` should store the odd numbers in list `L1` in the list `L2`. Update the program so it follows the given definition. Note, that the program should be written in a declarative way, i.e. every Horn clause should contain all necessary conditions.

```
?- store_odd_numbers([12,3,5,8,9],L).
L = [3,5,9] ? ;
no

store_odd_numbers([],List).
store_odd_numbers([A,B],[A]).
store_odd_numbers([E|Rest],List):-
      0 is E mod 1,
      store_odd_numbers(Rest,[E|List]).
```

Task 4 Work with text (11 points)

The letters å, ä and ö are quite frequently used in Swedish texts. How frequently are they used? Take a text and calculate the mean value of these letters in relation to all letters in the text. Both lower case and upper case letters should be taken into account. The mean value should be calculated in relation to other letters. This means that the characters that are not letters should not be counted. The program should write the mean value and the text according to the given format.

```
?- swedish_letters.
Mean value is 0.0796 of å, ä and ö in the text:
Du gamla, du fria, du fjällhöga nord. Du tysta, du glädjerika sköna!
Jag hälsar dig, vänaste land uppå jord, din sol, din himmel, dina ängder
gröna.
```

Ascii-code for upper case letters, except Å, Ä and Ö, are in the interval 65 – 90. Ascii-code for lower case letters, except å, ä and ö, are in the interval 97-122.

Assume that the predicate `ascii_code(Letter,Ascii)` is given and the text is stored in the predicate `text(T)`.

```
ascii_code('å',229).
ascii_code('ä',228).
ascii_code('ö',246).
ascii_code('Å',197).
ascii_code('Ä',196).
ascii_code('Ö',214).

% The first verse in the Swedish national anthem.
text('Du gamla, du fria, du fjällhöga nord
Du tysta, du glädjerika sköna!
Jag hälsar dig, vänaste land uppå jord,
din sol, din himmel, dina ängder gröna.').
```

Task 5 Work with a database (5 + 3 + 5 + 4 + 5 + 5 points)

The following database describes some typical Swedish dishes. Furthermore, it describes the ingredients that can be found at home for the moment.

```
:- dynamic dish/4.

% dish(Swedish_name,English_name,List_of_ingredients,Time_for_cooking)

dish('Köttbullar','Meat-balls',['minced meat', salt, pepper, egg,
   'brown dried', onion, water], time(30,minutes)).
dish('Biff a la Lindström','Beef a la Lindström',['minced meat', cream,
   onion, capers, 'pickled beetroot', salt, pepper], time(40,minutes)).
dish('Gravad lax','Raw spiced salmon',[salmon, salt, pepper, sugar,
   dill], time(2,days)).
dish('Köttfärssås','Mincedmeat sauce',[tomato, onion, 'minced
   meat', salt, pepper, sugar, paprika], time(40,minutes)).
dish('Dillkött','Mutton with dill sauce', [lamb, salt, pepper, onion,
   carrot, cream, dill, sugar, vinegar], time(100,minutes)).

% at_home(Ingredient)
at_home(water).
at_home(salt).
at_home(pepper).
at_home(egg).
at_home(paprika).
at_home(sugar).
at_home(vinegar).
at_home(onion).
```

a) Write a program that gives the Swedish name for a dish that contains a special ingredient, e.g. minced meat.

```
?- dishname('minced meat',SName).

SName = 'Köttbullar' ? ;
SName = 'Biff a la Lindström' ? ;
SName = 'Köttfärssås' ? ;
no
```

b) Give a list of all the dishes in Swedish that contains onion.

c) Make a shopping list for a special dish when the name is given in English. Do not put the ingredients that you have at home on the list.

```
?- shoppinglist('Raw spiced salmon', List).
List = [salmon, dill] ? ;
no
```

d) Present all the English names on dishes that can be cooked in 40 minutes, or less, in a list.

e) It should be possible for the user to change the list of ingredients related to a dish given in English by putting a new ingredient in the list and then store the new information in the database. The new ingredient should be included last in the list. The old information should of course be abolished.

```
?- new_ingredient(garlic,'Mincedmeat sauce'),listing(dish).
:
dish('Köttfärssås', 'Mincedmeat sauce', [tomato, onion,
   'minced meat', salt, pepper, sugar, paprika, garlic],
time(40,minutes)).
```

f) It should also be possible for the user to change a dish, given in English, by taking away an ingredient in the list of ingredients. The new information should be stored and the old taken away.

```
?- take_away_ingredient(garlic,'Mincedmeat sauce'),listing(dish).
:
dish('KöttfärssÅs', 'Mincedmeat sauce', [tomatoes, onion,
   'minced meat', salt, pepper, sugar, paprika], time(40,minutes)).
```

Task 6 Trace a Prolog program (8 + 3 + 3 + 4 points)

```
check([],1).
check([F|Rest],X):-
      l(F),!,
      check(F,A),
      check(Rest,B),
      X is A + B.
check([F|R],X):-
      check(R,X).

l([]).
l([_|_]).
```

a) If we ask the program the following, which value does the variable Answer get?

```
   ?- check([[a],b,[c,d,[]],e],Answer).
```

b) Explain in words what the program performs.

c) Is the program tail recursive? Justify your answer.

d) When writing tail recursive programs accumulator pairs are sometimes utilised. Described what an accumulator pair is and how it is used.

Task 7 Check the search space (5 + 3 + 6 points)

The program `union(L1,L2,UList)` should store all the elements in the lists `L1` and `L2` in `UList`, i.e. the elements that can be found in the input lists should be included in `UList`. But if an element can be found in both lists only one instance of these should be in `UList`.

a)

```
union([],L,L).
union([E|R],L2,L3):-
      member(E,L2),
      union(R,L2,L3).
union([E|R],L2,[E|L3]):-
      union(R,L2,L3).

member(E,[E|R]).
member(E,[F|R]):-
      member(E,R).
```

Give all alternative answers that the program can give on the following question:

```
?- union([a,b],[a,c,e,g],UList).
```

b) Rewrite the program in a declarative way so that only correct answers are generated when asking Prolog for alternative answers.

c) Another way to make sure that only correct answers are generated, than the solution in b), is to use cut (i.e. !). Rewrite the program with cut and describe whether you have used a red or green cut. Describe the characteristics for red and green cut and utilise an example in the description.

An extra task for fun…

GUESS A WORD
Write a program that gives the computer a word that a user then could guess. If the user guesses the correct word the user should get information about that and also get the opportunity to guess another word. If the word is not correct the program should present the letters that have the right positions in relations to the both words. Even the guessed letters that occur in the first word but not in the right position should be presented. Compare the game MasterMind.

Example 1:
```
The game leader gives its word
|: summer.
Guess the word
|: winter.
[*,*,*,*,e,r]
:
```

Example 2:
```
The game leader gives its word
|: eminent.
Guess the word
|: mammals.
[*,*,*,*,*,*,*]     Occurring letters: m
Guess the word
|: mothers.
[*,*,*,*,e,*,*]     Occurring letters: m t
:
```

In the last case the guess “mothers” has “e” in the right position compared to word “eminent”. The letters “m” and “t” can be found in the word but the user has not put them in the right positions.

This task is rather complicated and should be solved through divide-and-conquer. An example of a framework for solving the problem is given below and it can also be found in The Student portal.

```
% guess_a_word/0.
% starts the game, calls the predicate game(yes)

guess_a_word:-game(yes).


% game(+X) starts a new round if X is yes.
% If X is no the program ends.
% If X has another value a correct value is asked for.

game(no):-
  write('Thanks for a good game!'),nl.

game(yes):-
  read_Word(Word1),
  read_guess(Word2),
  play(Word1,Word2),
  write('Once more?'),nl,
  read(Answer),
  game(Answer).

game(X):-
  write('Unknown alternative, answer yes or no'),nl,
  read(Answer),
  game(Answer).


% read_Word(-List) reads the Word that the game leader gives and that
% the player should guess. Except the reading the predicate translates
% the atom to a list of letters

read_word(List):-
  write('The game leader gives its word'),nl,
  read(Word),
  name(Word,CharList),
  letters(CharList,List).


% letters(+AsciList,-LetterList) translates an ascii list
% to a list with the corresponding letters.
% This recursive program utilises the built-in predicate name/2


letters(X,Y):-.............
% read_guess(-List) reads the word the player guesses
% and translates it to a list of letters
read_guess(List):- .............
```

```
% play(+X,+Y) if X and Y are of equal length, play succeeds directly.
% If the lengths differ X and Y should be compared, the comparison
% presented and a new try should be performed.

play(Word,Word):-
  write('Correct!'),nl.

play(Word1,Word2):-
  Word1\==Word2,
  same_placing(Word1,Word2,Starlist),
  included_but_not_same_placing(Word1,Word2,List),
  output(Starlist,List),
  read_guess(NewWord),
  play(Word1,NewWord).


% same_placing(+Word1,+Word2,-Starlist)
% checks for the letters that have the right placing.
% The predicate produces a list with stars for positions where the letters are
% not the correct ones. The correct placed letters is included.
% Exemple: The guess is winter and the correct word is summer
% the Starlist will be: [*,*,*,*,e,r]

same_placing(Word1,Word2,Starlist):-..........


% included_but_not_same_placing(+Word1,+Word2,-List)
% will check the guessed letters that can be found in the
% correct word but not in the right position

included_but_not_same_placing(Word1,Word2,List):- .........


% output(+Starlist,+List) should write the message to
% player presenting the result.

output(Starlist,[]):-.................
```