

Assignment # 5 (Due: Thursday, Dec. 3rd 2009, 6:00pm sharp)

Details: - Programming in Prolog. This is a short assignment, which weights 8%.
- Note that according to school policy, no assignment can be accepted after the last day of lectures,

The last day of lecture in computer science dept. is Dec. 4th 2009. Hence, please make sure you submit by Dec. 4th 2009, 6:00pm sharp.

Silent Policy

A silent policy will take effect 24 hours before this assignment is due. This means that no question about this assignment will be answered, whether it is asked on the newsgroup, by email, or in person.

Handing in this Assignment

You should submit your work electronically. To submit files electronically, use the CDF secure website:

<https://www.cdf.toronto.edu/students>

or use the CDF **submit** command. Type **man submit** for more information. You should submit the following files:

a5.pl The Prolog code for your solutions.

testing.txt Your testing document (see below).

coverpage.txt An electronic cover page for your assignment.

Warning: You must follow the guidelines for electronic submission, available on your course website for assignments. Marks will be deducted for incorrect submission.

Testing

For each function that you write, you must describe the testing strategy that you used. This should include the test cases that you designed for your function, what output your function returned, and an explanation of why the test case and output are significant in verifying the correctness of the function. Read the following for a discussion of good testing practices:

<http://www.cs.toronto.edu/~yilan/324f09/testing.pdf>

The test file must be named as **testing.txt**. Only plain text submissions will be marked.

IMPORTANT NOTE

Since we will test your code electronically, **you must:**

1. Make certain that your code runs in SWI Prolog on CDF.
2. Comment out any partial solutions, or code that contains errors. If your submission contains errors that prevent it from running on CDF, it will receive no marks for correctness.
3. Use the exact predicate names specified.
4. Use the exact file name specified.

A5 Bulletin Board

Important corrections (hopefully few or none) and clarifications to the assignment will be posted on the Assignment webpage, linked from your CSC324 home page. You are also responsible for monitoring the CSC324 bulletin board.

How you will be marked

Code will be marked with respect to correctness, style and documentation. For style and documentation guidelines, please see the assignment website.

Assignment # 5

This assignment is a warm-up to get you used to programming in Prolog and to thinking as a logic programmer. You may use helper predicates as needed in defining these predicates. You may **not** use any of the following:

```
; ("or")      -> ("if-then")
```

or any other special operators in Prolog (which generally subvert the pure logic programming paradigm) in this assignment. If you are in doubt about what's allowed, check the Assignment Clarification page or ask on the Bulletin Board; if you haven't seen a certain notation in class or tutorial, please check with me.

Prolog Notational Conventions: Predicate and Mode Spec

We shall use the following notation when *referring* to Prolog predicates: Predicates in Prolog are distinguished by their name and their arity. The notation name/arity is therefore used when it is necessary to refer to a predicate unambiguously; e.g., `append/3` specifies the predicate named "append" that takes 3 arguments. Sometimes, we may be interested in specifying how specific predicates are meant to be used. To that end, we shall present a predicate's usage with a *mode spec* which has the form: `name(arg_1, ..., arg_n)`, where each `arg_i` denotes how that argument should be instantiated when a goal to `name/n` is called. `arg_i` has one of the following forms:

+ArgName This argument should be instantiated to a non-variable term.

-ArgName This argument should be uninstantiated.

?ArgName This argument may or may not be instantiated.

For example, `delete(+List,?Elem,?NewList)` states that, when using `delete/3`, the first argument should be instantiated whereas the second and third arguments may or may not be instantiated. Note that these Prolog notational conventions provide a convenient way to *specify* Prolog predicates and their usage. They do not represent in any way the form of your actual code. For instance, when defining a predicate **subs/4** in Question 1 below, do not use "+X", "+Y", "+L1" or "?L2" as your arguments in your code.

Do's and Don'ts

1. You should not use any special Prolog/SWI-Prolog features like `assert`, `retract`, `arg`, Put differently, the only predicates you may use apart from the ones you implement yourself are `not/1`, `append/3`, `member/2`, `length/2`, 'is', arithmetic symbols and the different matching/equal symbols we mentioned in class. Of course you may also use output predicates like `writeln/1` for testing and debugging, but these have to be removed prior to submission. If there is another built-in predicate you wish to use, please consult the instructor before using it.
2. Avoid *singleton* variables! A singleton variable is a variable that only occurs once in the arguments or the body of a predicate and is thus useless. For example in the following two definitions:

```
doit(X,Y,Z) :- Y is X * 2.  
doitagain(X,Y) :- Z = doitmatter, Y is X * 2.
```

`Z` is a singleton variable and should be prefixed by an underscore (`_Z`) or removed entirely if possible. Note that it usually won't be possible to simply remove a singleton argument as the arity of the predicate is often fixed:

```
member(X, [X|T]) .  
member(X, [_|T1]) :-member(X, T1) .
```

Here in the first definition `T` is singleton but cannot be removed. Therefore we should replace it with '`_T`' or just '`_`'. Although not harmful, it is useful not to have any singleton variables to keep the code easy. SWI-Prolog will point out any singleton variables you have. This is very useful information because it often helps you finding typos, a common source of bugs in Prolog. For instance if we want to increase a number we could write:

```
inc(FirstNr,Result) :- Result is Firstnr +1.
```

Here both `FirstNr` and `Fristnr` are singletons and SWI-Prolog will tell you so, which in turn will make you realize that you have a typo.

Marks will be deducted for each singleton you have in your code!

3. You have to use the exact predicate names requested in the questions, but you may define helper predicates when necessary.

Question 1. [5 points]

Write a predicate **rmlast(?X,?Y)** that succeeds if X and Y are lists, and Y is the same as X except that X's last element is not present in Y. Neither X nor Y is required to be instantiated, and when backtracking after success (i.e., typing ";" asking for additional possible solutions) you should produce every correct answer **exactly once**. You may assume that X is not empty. Save your program in a5.pl. Test your predicate and provide some sample queries.

Question 2. [10 points]

Write a predicate **secondlargest(+List,?Val)** that succeeds if List is a list of numbers and Val is equal to the second-largest element of List. If there is a tie for first place, the second-largest number is the same as the largest. You may require List to be instantiated and at least has two elements. If more than one entry in List has the second-largest value, you should nevertheless produce the correct answer **just once and terminate the query right away without allowing backtracking**. Save your program in a5.pl. Test your predicate and provide some sample queries.

Question 3. [20 points] (Logic Puzzle)

Donald and Daisy Duck took their nephews aged 4, 5 and 6 on an outing. Each boy wore a tee-shirt with a different design on it and of a different colour. You are also given the following information:

- (1) Huey is younger than the boy in the green tee-shirt.
- (2) The five year-old wore the tee-shirt with the camel design.
- (3) Dewey's tee-shirt was yellow.
- (4) Louie's tee-shirt bore the giraffe design.
- (5) The panda design was not featured on the white tee-shirt.

Now write a Prolog program, to conclude the age of each boy by following the steps below.

- (a) write the facts in the above problem using predicates:
age(L) - L is the list of all available ages of the boys
color(L) - L is the list of all available colors of the tee-shirts
design(L) - L is the list of all available design of the tee-shirt.

(b) write a rule for **switchOrder(+L,?New)**, so that New is a list that has the same elements as those in L, but may or may not be in a different order.

(c) write a rule for predicate **cond1(+AgeList,+DesignList)**, assuming that AgeList represents a possible solution to the age of Huey, Dewey and Louie in order and DesignList represents a possible solution to the design of the tee-shirts Huey, Dewey and

Louie were wearing in order. Moreover, `cond1(+AgeList, +DesignList)` is true if the five year-old wore the tee-shirt with the camel design.

(d) solve the puzzle by define a rule for predicate
`solve([?HueyAge, ?DeweyAge, ?LouieAge])`

so that when query `"?- solve([HueyAge, DeweyAge, LouieAge])."` is asked, the solution of the puzzle will be displayed. The general idea is to use the facts in (a) and the predicate in (b) to generate the possible solutions (as lists) for the ages, color and design of tee-shirts for Huey, Dewey and Louie (in order), and make sure these possible solutions satisfies all the conditions described in the puzzle problem. For instance, (c) is an example of how you may define your helper predicates to describe the conditions in the puzzle.

Save your program in `a5.pl`. Test your predicate and provide some sample queries.

Question 4. [10 points]

A **heap** is a data structure created using a binary tree. It can be seen as a binary tree with two additional constraints:

- *The shape property: the tree is an almost complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.*
- *The heap property: each node is greater than or equal to each of its children according to some comparison predicate which is fixed for the entire data structure.*

Note that a heap is not a binary search tree, and that it is unrelated to the kind of heap used in dynamic memory allocation. (Heaps are used in "heap sort", a fast sorting algorithm.)

Write a predicate **`heap(+Tree)`** that succeeds if `Tree` is a heap. You may assume that `Tree` must be instantiated, and the data contained in `Tree` must be integers.

For this exercise, use functional term `node(K,L,R)` to represent a tree with key `K` (an integer), left subtree `L` and right subtree `R`. The atom `"empty"` represents an empty binary tree.

As an example, the almost complete binary tree

```
node(5, node(4, empty, empty), empty)
```

is a heap, because the root's data 5 is greater than the data 4 in the root's only child. Save your program in `a5.pl`. Test your predicate and provide some sample queries.

Question 5. [15 points]

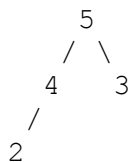
Repeat Exercise 4, but with a different tree representation: a list.

An almost complete binary tree is a binary tree in which every level is full except possibly the bottom level, where there are only leaves. If the bottom level is not full, the nodes of that level are filled from left to right.

An almost complete binary tree can be represented as an array, or in Prolog as a list. In this list, the first element (at index 1) is the root, and the children of element N are in the nodes at indices 2N and 2N+1.

Note that the preceding paragraph uses indices starting at 1, not 0, because it's easier to describe the tree representation that way.

Define a **heap_q5(+Tree)** predicate is true iff Tree is a heap tree, and you may assume that Tree is a valid complete binary tree containing integer data. With the list representation, that just means you can assume Tree is a list of integers. For example, the list [5, 4, 3, 2] is a complete binary tree with root 5. The root's children are 4 and 3, and 4's child is 2. This tree is in fact also a heap.



You cannot assume there is any special integer that represents a missing node: you know you're at a leaf when both the children would be past the end of the list.

Examples:

```
?- heap_q5([5, 4, 3, 2]).
true
?- heap_q5([7, 6, -1, -3, 5]).
true
?- heap_q5([]).
true
```

Save your program in a5.pl. Test your predicate and provide some sample queries.