F28PL Coursework 3, Prolog
(deadline: 11th December 2020)

Fork this project

https://gitlab-student.macs.hw.ac.uk/f28pl-2020-21/f28pl-2020-21-prolog-coursework

into your own namespace.

Important notes

(1) Code should be clearly written and should include a brief explanation in English explaining its design.
(2) For all questions, include testing in comments, so your marker can load this file as a database then cut-and-paste any testing into the command line.
(3) Testing on GitLab will NOT be provided for Prolog. Your own test will in this case be marked — note this is unlike the Python and OCaml coursework.
(4) Submit by pushing your code in the file ***pl_cw.pl*** to the GitLab server. Only code that has been pushed to **your fork** of the **f28pl-2020-21-prolog-coursework** before the deadline will be marked.
We are not using Vision for coursework submission.
(5) You cannot use library functions if they make the question trivial.
(6) You can write your own helper functions, if convenient.
(7) Code should be clearly written and laid out and should include a brief explanation in English explaining the design of your code.
(8) Consistent with the principle that *code is written for humans to read* in the first instance, and for computers to execute only in the second instance, marks will be awarded for *style and clarity.*
(9) You may use functions defined in answers to previous questions, in later questions.
(10) Use the ***swipl*** interactive application alongside an editor for the source code when developing your solution and use **git** to push your commits to the GitLab repository.
(11) Code must be valid Prolog. Your marker will expect you to deliver code that compiles. **Code that cannot even compile, may score zero marks.**
(12) If you are in difficulty with this coursework then be in touch with the lecturer ASAP to discuss getting additional support. We can only help you if you ask.

1. **Complex number arithmetic (yes, again!)**

The **complex numbers** are explained here (and elsewhere):

http://www.mathsisfun.com/algebra/complex-number-multiply.html

Represent a complex integer as a two-element list of integers, so `[4,5]` represents 4+5j.

Write Prolog predicates

```
cadd/3
cmult/3
```

representing complex integer addition and multiplication. Thus for instance,

```
cadd([X1,X2],[Y1,Y2],[Z1,Z2])
```

succeeds if and only if `Z1=X1+Y1` and `Z2=X2+Y2`.

Note that complex number multiplication is not just like complex number addition. Check the link and read the definition.

2. **Sequence arithmetic (yes, again ⏮)**

An **integer sequence** is a list of integers. Write a Prolog predicate

```
seqadd/3
```

such that `seqadd(Xs,Ys,Zs)` succeeds when `Xs` and `Ys` are lists of integers of the same length and `Zs` is their sequence sum.

3. **Essay question**

3a. Explain what **backtracking** has to do with Prolog. You might find this webpage helpful:

https://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages/search.html

3b. What is Cut in prolog and what does it have to do with backtracking? Explain your answer by giving examples of Cut used in at least one prolog rule, and explain how it affects the execution/resolution process.

Submit the essay as Prolog comments where the space is provided.

4. Generator

Write a database for a predicate `cycleoflife/1` such that the query

```
cycleoflife(X)
```

returns the instantiations

```
X = eat ;
X = sleep ;
X = code ;
X = eat ;
X = sleep ;
X = code ;
...
```

in an endless cycle.

(This question has a beautiful and simple answer. If you find yourself writing lines and lines of complex code, there's probably something amiss.)