CSC3403 S1 2022 Assignment 3

Logic Programming Due Date: 30 May 2022

Objective

This assignment aligns with Course Learning Outcome 3. It assesses your ability to:

- Analyse and compare different language paradigms, with a particular emphasis on logic programming languages.

Marks

There are **two (2)** tasks each comprising several parts in Assignment 3. There are **20 marks** and Assignment 3 comprises **20%** of your final mark.

Submission

- Submit a single plain text file (extension `.txt`) containing all definitions. Please use your user id as the name of the file; e.g. `u12345678.txt`.
- The text file must only contain valid Prolog code and comments.
- **Use the specified predicate names and arity as your code will be tested by a Prolog function expecting that predicate name**.
  - You may however write any 'helper' functions that you need to be able to solve any of the following problems.
- Final submission is via the submission link for Assignment 3 on the CSC3403 Study Desk.
- Unless stated otherwise **do not use library functions that are not in the Prolog standard prelude**
- You must use techniques **which have been presented in the course**. Using sophisticated code and methods which have not been presented will attract penalties.
- You can use SWI Prolog on your own systems or use the provided resources on the USQ Turbo system used in the course Workshops for your code.

Late Submission of Assignments[1]

Students can apply for an extension of time to submit an assignment at any time **up to** the deadline. Students are advised to make a request for an extension as soon as their need becomes apparent. Delay in making a request involves the risk of losing marks if the request is refused.

Extensions are usually granted only in cases of Compassionate and Compelling Circumstances in accordance with the assessment of Compassionate and Compelling Circumstances Procedure. Generally, extensions will be limited to a maximum of five University Business Days. A Student requiring an extension for a period of time in excess of this should consider applying for a Deferred Assessment as per Section 4.4 of the assessment procedure.

Applications for extensions must be made through your **Student Centre**. An Assignment submitted after the deadline without an approved extension of time will be penalised. The penalty for late submission without a pre-approved extension is a reduction by 5% of the maximum mark applicable for the assignment, for **each *calendar* day** or part day that the assignment is late. An assignment submitted more than **one (1) week** after the deadline will have a mark of zero recorded for that assignment.

The Examiner may refuse to accept assignments for assessment purposes after marked assignments and/or feedback have been released.

Non-submission of Assignments and Passing Grades[2]

To be assured of receiving a passing grade a student must obtain at least 50% of the total weighted marks available for the course and have satisfied any Secondary Hurdles (if applicable).

Supplementary assessment may be offered where a student has undertaken all of the required summative assessment items and has passed the Primary Hurdle but failed to satisfy the Secondary Hurdle (Supervised), or has satisfied the Secondary Hurdle (Supervised) but failed to achieve a passing Final Grade by 5% or less of the total weighted Marks.

To be awarded a passing grade for a supplementary assessment item (if applicable), a student must achieve at least 50% of the available marks for the supplementary assessment item as per the Assessment Procedure (point 4.4.2).

[1] See http://policy.usq.edu.au/documents.php?id=14749PL for the full assessment regulations.
[2] See the University Assessment Procedure policy document

The offer of Supplementary Assessment normally will only be made if the Student has undertaken all possible Summative Assessment Items for the Course (i.e. the assignments).

Student Responsibilities

The Assessment Procedure Section 4.2.2, also outlines the following student responsibilities:

- If requested, Students must be capable of providing a copy of Assignments submitted. Copies should be dispatched to the University within 24 hours of receipt of a request being made.
- Students are responsible for submitting the correct Assignment.
- Assignment submissions must contain evidence of student effort to address the requirements of the Assignment. In the absence of evidence of Student effort to address the requirements of the assignment, no Mark will be recorded for that Assessment Item.
- A Student may re-submit an Assignment at any time up to the deadline. A request to re-submit after the deadline is dealt with in accordance with section 4.4 'Deferred, Supplementary and Varied Assessment and Special Consideration' of these procedures.

Academic Integrity (AI)

Academic misconduct is unacceptable and includes plagiarism, collusion and cheating:

- *plagiarism*: involves the use of another person's work without full and clear referencing and acknowledgement;
- *cheating*: involves presenting another person's work as your own — this includes ***contract cheating***.
- *collusion*: is a specific type of cheating, that occurs when two or more students fail to abide by directions from the examiner regarding the permitted level of collaboration on an assessment.

All are seen by the University as acts of misconduct for which you can be penalised. For further details go to the Library's site on Academic Integrity.

Task 1 (12 marks)

Topics:	Logic programming; recursive programming.
Submission:	Part of single `.txt` document containing all of your Prolog code and comments for the assignment. **Do not include any facts or database elements in your submission**
Notes:	You must use the predicate names and arity as specified. You may write helper predicates that the specified function calls. Not all solutions *necessarily* require a recursive function. However, if the task requests it specifically, the function must be recursive.

Background

Family trees are a common example and teaching tool in logic programming. This is because there are relationships which are easily understood by most.

In our Programming exercise, we are going to examine a particular family tree, the Wellesleys, the family of the Dukes of Wellington.

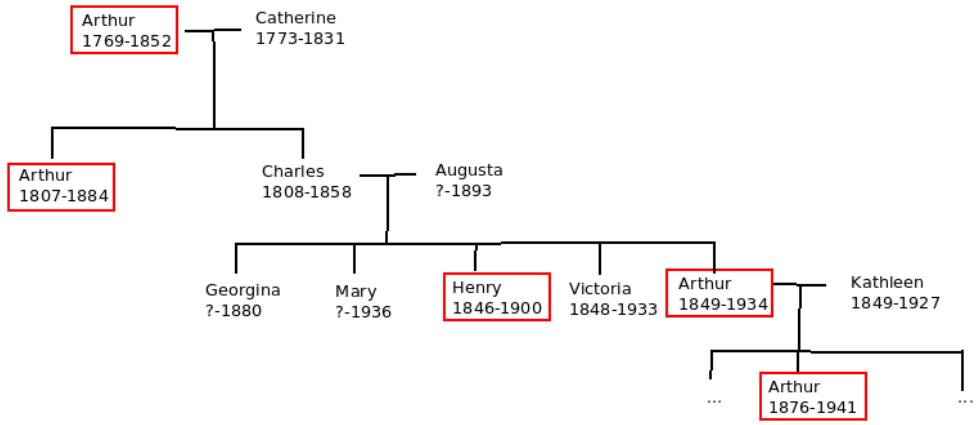

Figure 1: Family tree of the Dukes of Wellington. The Dukes are in red boxes.

The Dukedom is passed down via male-line primogeniture. In other words, to the nearest living male relative. The First Duke (Arthur, 1769-1852) was succeeded by his eldest son. However, the Second Duke died childless, and his brother had pre-deceased him. Hence, the Third Duke was Henry (1846-1900), the second Duke's nephew. Henry also dies childless, and the Dukedom went to his brother Arthur.
On page 8 is a code that, given the fact of a Duke, it will return the details of the next Duke.

Example output (SWI-Prolog):

```
?- nextduke(wellesley(arthur,m,dates(1769,1852)),NEXTDUKE).
NEXTDUKE = wellesley(arthur, m, dates(1807, 1884)).

?- nextduke(wellesley(arthur,m,dates(1807,1884)),NEXTDUKE).
NEXTDUKE = wellesley(henry, m, dates(1846, 1900)).

?- nextduke(wellesley(henry,m,dates(1846,1900)),NEXTDUKE).
NEXTDUKE = wellesley(arthur, m, dates(1849, 1934)).

?- nextduke(wellesley(arthur,m,dates(1849,1934)),NEXTDUKE).
NEXTDUKE = wellesley(arthur, m, dates(1876, 1941)).
```

The code on page 8 calls several functions. Your task will be to write the functions as directed. ***You do not need to get the entire code working;*** each function can be tested on its own using various facts from the database.

Task 1.1 (2 marks) Write a recursive predicate:

```
maledescend(FATHER,DESCEND)
```

that lists all male descendants of the given `FATHER`.

- You may use helper functions; recursion may occur in calling or called functions or both
- **Do not submit the facts or code from page 8 in your submission file.**

Example output (SWI-Prolog):

```
?- maledescend(wellesley(charles,m,dates(1808,1858)),D).
D = wellesley(henry, m, dates(1846, 1900)) ;
D = wellesley(arthur, m, dates(1849, 1934)) ;
D = wellesley(arthur, m, dates(1876, 1941)) ;
false.

?- maledescend(wellesley(arthur,m,dates(1807,1884)),D).
false.
```

Task 1.2 (2 marks) Write a predicate:

```
children(FATHER,CHILDREN)
```

tha produces a **list** of a `FATHER`'s `CHILDREN`.

- You may use helper functions.

Example output (SWI-Prolog):

```
?- children(wellesley(charles, m, dates(1808,1858)),C).
C = [wellesley(georgina, f, dates(unk, 1880)), wellesley(mary, f, dates(unk, 1936)),
    wellesley(henry, m, dates(1846, 1900)), wellesley(victoria, f, dates(1848, 1933)),
    wellesley(arthur, m, dates(1849, 1934))].

?- children(wellesley(henry, m, dates(1846,1900)),C).
C = [].
```

Task 1.3 (2 marks) Write a recursive predicate:

```
sons(CHILDREN,SONS)
```

that takes a list of `CHILDREN` and produces a list of `SONS`.

- You may use helper functions.

Example output (SWI-Prolog):

```
?- sons([wellesley(georgina, f, dates(unk, 1880)), wellesley(mary, f, dates(unk, 1936)),
   wellesley(henry, m, dates(1846, 1900)), wellesley(victoria, f, dates(1848, 1933)),
   wellesley(arthur, m, dates(1849, 1934))],S).
S = [wellesley(henry, m, dates(1846, 1900)), wellesley(arthur, m, dates(1849, 1934))]
```

Task 1.4 (2 marks) In the code on page 8, there is a function:

```
eldestson(DUKE,HEIR) :-
        children(DUKE, CHILDREN), sons(CHILDREN,SONS), findeldest(SONS,HEIR).
```

Write the recursive function:

`findeldest(PEOPLE,ELDEST)`

that takes a list of `PEOPLE` and returns the `ELDEST` person (i.e. the person born first). Thus passing a list of sons would produce the eldest son!

- You may use helper functions.

Example output (SWI-Prolog):

```
?- findeldest([wellesley(henry, m, dates(1846, 1900)),
   wellesley(arthur, m, dates(1849, 1934))],E).
E = wellesley(henry, m, dates(1846, 1900)).
```

Task 1.5 (2 marks) Write the predicate:

`brother(X, Y)`

such that `Y` is the brother (i.e. male sibling) of `X`.

- You may use helper functions.

Example output (SWI-Prolog):

```
?- brother(wellesley(charles,m,dates(1808,1858)),B).
B = wellesley(arthur, m, dates(1807, 1884)) ;
false.

?- brother(wellesley(georgina,f,dates(unk,1880)),B).
B = wellesley(henry, m, dates(1846, 1900)) ;
B = wellesley(arthur, m, dates(1849, 1934)) ;
false.

?- brother(X,wellesley(henry, m, dates(1846, 1900))).
X = wellesley(georgina, f, dates(unk, 1880)) ;
X = wellesley(mary, f, dates(unk, 1936)) ;
X = wellesley(victoria, f, dates(1848, 1933)) ;
X = wellesley(arthur, m, dates(1849, 1934)) ;
false.
```

Task 1.6 (2 marks) Up to two marks will be awarded for comments in your predicates for Tasks 1.1 through Task 1.5. Comments which show your understanding of each predicate rule will be awarded marks.

Facts and Predicates for Task 1

```
% Facts
% wellesley(NAME,GENDER,dates(BORN,DIED)
%
wellesley(arthur,m,dates(1769,1852)).    % First Duke of Wellington
wellesley(catherine,f,dates(1773,1831)).
wellesley(arthur,m,dates(1807,1884)).    % Second Duke of Wellington
wellesley(charles,m,dates(1808,1858)).
wellesley(augusta,f,dates(unk,1893)).
wellesley(georgina,f,dates(unk,1880)).
wellesley(mary,f,dates(unk,1936)).
wellesley(henry,m,dates(1846,1900)).     % Third Duke of Wellington
wellesley(victoria,f,dates(1848,1933)).
wellesley(arthur,m,dates(1849,1934)).    % Fourth Duke of Wellington
wellesley(kathleen,f,dates(1849,1927)).
wellesley(arthur,m,dates(1876,1941)).    % Fifth Duke of Wellington
%
% Marriage Relationships
%
married(wellesley(arthur,m,dates(1769,1852)), wellesley(catherine,f,dates(1773,1831))).
married(wellesley(charles,m,dates(1808,1858)), wellesley(augusta,f,dates(unk,1893))).
married(wellesley(arthur,m,dates(1849,1934)), wellesley(kathleen,f,dates(1849,1927))).
%
% Father -> Child Relationships
%
father(wellesley(arthur,m,dates(1769,1852)), wellesley(arthur,m,dates(1807,1884))).
father(wellesley(arthur,m,dates(1769,1852)), wellesley(charles,m,dates(1808,1858))).
father(wellesley(charles,m,dates(1808,1858)), wellesley(georgina,f,dates(unk,1880))).
father(wellesley(charles,m,dates(1808,1858)), wellesley(mary,f,dates(unk,1936))).
father(wellesley(charles,m,dates(1808,1858)), wellesley(henry,m,dates(1846,1900))).
father(wellesley(charles,m,dates(1808,1858)), wellesley(victoria,f,dates(1848,1933))).
father(wellesley(charles,m,dates(1808,1858)), wellesley(arthur,m,dates(1849,1934))).
father(wellesley(arthur,m,dates(1849,1934)), wellesley(arthur,m,dates(1876,1941))).
%
% To find the next Duke, by male primogeniture:
%    - Eldest son of the Duke (if alive)
%    - Duke's Brother (if alive; in this case, each Duke has only one brother, otherwise eldest brother)
%    - Eldest son of Duke's brother (if alive)
%
nextduke(wellesley(DNAME,DG,dates(DB,DD)),wellesley(NAME,G,dates(B,D))) :-
             eldestson(wellesley(DNAME,DG,dates(DB,DD)),wellesley(NAME,G,dates(B,D))),
                        B < DD,
                        D > DD, !.
nextduke(wellesley(DNAME,DG,dates(DB,DD)),wellesley(NAME,G,dates(B,D))) :-
              brother(wellesley(DNAME,DG,dates(DB,DD)),wellesley(NAME,G,dates(B,D))),
                        B < DD,
                        D > DD, !.
nextduke(wellesley(DNAME,DG,dates(DB,DD)),wellesley(NAME,G,dates(B,D))) :-
            brother(wellesley(DNAME,DG,dates(DB,DD)),BROTHER),
                       eldestson(BROTHER,wellesley(NAME,G,dates(B,D))),
                       B < DD,
                       D > DD, !.

eldestson(DUKE,HEIR) :-
         children(DUKE, CHILDREN), sons(CHILDREN,SONS), findeldest(SONS,HEIR).
```

End of Task 1

Task 2 (8 marks)

Topics:	Logic programming; recursive programming.
Submission:	Part of single `.txt` document containing all of your Prolog code and comments for the assignment. **Do not include any facts or database elements in your submission**
Notes:	You must use the predicate names and arity as specified. You may write helper predicates that the specified function calls. Not all solutions *necessarily* require a recursive function. However, if the task requests it specifically, the function must be recursive.

Background

On page 12 is part of a code that implements a simplified version of the popular word game Wordle[3] followed by example runs of the game.

The goal of the game is to guess a five-letter word in six tries. In our simplified game, the word contains no duplicate letters. Additionally, there is no checking that the guess is a dictionary word as in the real game.

When a guess is made, an indication is made for each letter:

- Green (`'g'`): Letter is in the target word and in the correct position in the target word
- Yellow (`'y'`): Letter is in the target word but in the wrong position in the target word
- Black (`'b'`): Letter is not in target word

For example, if the target word is `'blues'` and the guess is `'boast'`, the output from the game is `['g','b','b','y','b']`.
To win the game the player needs to get "all green" (`['g','g','g','g','g']`) in six turns.
Your task is to write some of the required predicates as specified below. **You do not need to make the entire code work. Each function you write can be tested independently!**

[3] https://www.nytimes.com/games/wordle/index.html

Task 2.1 (1 mark) Write the recursive predicate:

`allgreen(LIST)`

that checks if the player has received "all green".

- You may use helper functions but you must use recursion
- You do not need to check that the input is 5 elements long, the program will only pass a 5-element list.

Example Output (SWI-Prolog):

```
?- allgreen(['g','g','g','g','g']).
true.

?- allgreen(['g','b','y','g','g']).
false.
```

Task 2.2 (3 marks) Not shown in the code is a predicate:

```
inpos(L1,[X|L2]) :- findindex([X|L2],I),
    charbyindex(I,L1,Y),
    X == Y.
```

that is used to see if a given element (`X`) is in the correct position in the target word (the list, `L1`).
Write a recursive predicate:

`charbyindex(INDEX,LIST,ELEMENT)`

that returns the `ELEMENT` in position `INDEX` in the given `LIST`.

- You may use helper functions but you must use recursion
- You do not need to check that the input is 5 elements long, the program will only pass a 5-element list and this predicate should work on lists of any length.

Example Output (SWI-Prolog):

```
?- charbyindex(0,['a','b','c'],E).
E = a.

?- charbyindex(1,['a','b','c'],E).
E = b.

?- charbyindex(2,['a','b','c'],E).
E = c.
```

```
?- charbyindex(4,['a','b','c'],E).
false.

?- charbyindex(2,['a','b','c'],b).
false.

?- charbyindex(2,['a','b','c'],c).
true.
```

Task 2.3 (4 marks) For this question, you likely can not run it as some of the required program code is missing on page 8! However, given the code below, you should be able to infer the answer.
The recursive predicate

```
do_guess(TARGETLIST,GUESSLIST,RESPONSELIST)
```

builds the response list (e.g. `['g','y','b','g','g']`). The code is shown below, but the first two rules are missing:

```
1 do_guess(    ) :- .
2 do_guess(    ) :- .
3 do_guess(TLIST, [X|GL], ['y'|RL]) :- member(X,TLIST),
4      not(inpos(TLIST,[X|GL])), !,
5      do_guess(TLIST,GL,RL).
6 do_guess(TLIST, [X|GL], ['g'|RL]) :- member(X,TLIST),
7      inpos(TLIST,[X|GL]), !,
8      do_guess(TLIST,GL,RL).
```

Write the missing rules so that the code will work correctly *and* write comments explaining the operation of each rule (including existing rules shown and the two you write)!

Example Output (SWI-Prolog):

```
?- do_guess([a, b, c, d, e],[c,b,d,g,h],R).
R = [y, g, y, b, b].
```

End of Task 2

Wordle Code for Task 2:

```
%% PROLOG Wordle!
%%
%% Five letter words with no duplicate letters allowed
%%
%% Enter the puzzle number, then make up to 6 guesses.
%%
%% 'b' = letter is no in the word
%% 'y' = letter in the word be not that location
%% 'g' = letter in the word in correct location
%%
%%

wordle :- write('Enter puzzle number: '),
      read(PUZNO),
      write('Turn 1 - Enter your guess: '),
      read(GUESS),
      process(GUESS,PUZNO,1).

wordle(TURN,PUZNO) :- TURN == 7,
      target(PUZNO,WORD),
      write('Sorry! - The word was '),
      write(WORD), nl,
      process(stop, 0, TURN).

wordle(TURN,PUZNO) :- write('Turn '),
      write(TURN), write(' - Enter your guess: '),
      read(GUESS),
      process(GUESS,PUZNO,TURN).

process(stop,_,_) :- !.
process(GUESS,PUZNO,_) :- wordle_guess(PUZNO,GUESS,RESULT),
      allgreen(RESULT),
      write(RESULT),nl, write('Got it!'), nl, !.

process(GUESS,PUZNO,TURN) :- string_chars(GUESS, GLIST),
      length(GLIST,LEN), LEN =\= 5,
      write('Invalid - guess must be 5 characters long!'), nl, !, wordle(TURN,PUZNO).

process(GUESS,PUZNO,TURN) :- string_chars(GUESS, GLIST),
      not(no_dups(GLIST)),
      write('Invalid - guess must no duplicates!'), nl, !, wordle(TURN,PUZNO).

process(GUESS,PUZNO,TURN) :- wordle_guess(PUZNO,GUESS,RESULT),
      write(RESULT),nl, NEXTTURN is TURN+1,
      wordle(NEXTTURN,PUZNO).

wordle_guess( PUZNO, GUESS , RESULT ) :-
      wordle_target(PUZNO, TLIST),
          string_chars(GUESS, GLIST),
      do_guess(TLIST, GLIST, RESULT).

wordle_target(PUZNO, LIST) :- target(PUZNO,WORD),
      string_chars( WORD, LIST ).
```

Example output (SWI-Prolog):

```
?- wordle.
Enter puzzle number: 114.
Turn 1 - Enter your guess: |: peach.
[g,y,y,b,b]
Turn 2 - Enter your guess: |: prams.
[g,b,y,b,b]
Turn 3 - Enter your guess: |: pleas.
[g,g,g,g,b]
Turn 4 - Enter your guess: |: plead.
[g,g,g,g,g]
Got it!
true.
```

```
?- wordle.
Enter puzzle number: 112.
Turn 1 - Enter your guess: |: frees.
Invalid - guess must no duplicates!
Turn 1 - Enter your guess: |: fres.
Invalid - guess must be 5 characters long!
Turn 1 - Enter your guess: |: shear.
[y,y,y,y,y]
Turn 2 - Enter your guess: |: heart.
[g,g,g,g,b]
Turn 3 - Enter your guess: |: hears.
[g,g,g,g,g]
Got it!
true.

?- wordle.
Enter puzzle number: 113.
Turn 1 - Enter your guess: |: prize.
[b,g,g,b,y]
Turn 2 - Enter your guess: |: pairs.
[b,b,g,y,g]
Turn 3 - Enter your guess: |: lairs.
[b,b,g,y,g]
Turn 4 - Enter your guess: |: liars.
[b,y,b,y,g]
Turn 5 - Enter your guess: |: byers.
[b,b,y,y,g]
Turn 6 - Enter your guess: |: liers.
[b,y,y,y,g]
Sorry! - The word was fries
true .
```

End of Assignment 3

You should have one (1) `.txt` file containing only valid Prolog comments and code.

These files should then be uploaded using the Assignment 3 submission link on the Study Desk.

This assignment specification was typeset using LaTeX