Coursework Project 3

Symbolic and Declarative Computing & AI (COS5012B)

Prolog and Database Manipulations

November 29, 2016

This is your third coursework sheet in SDC/AI, session 2016-2017. The coursework tasks formulated on these pages define your Prolog programming project. Itcomprises ten subtasks. The maximum number of marks achievable for every subtask is 10; therefore, in total you can achieve 100 marks. *marking advice*

You are advised to follow the guidance given below. Try to work through the sheet in a systematic fashion (i.e. from beginning to end with as much coverage as possible). Partial solutions will also attract marks if the idea(s) involved show promise or substance.

Individual working and submission are required. **Number the pages in the style of this document.** Typewritten/word processed documents are a *must* in this case as this is the evidence of your software working as required.

You may be asked to demonstrate and explain your submission in person to me and/or the demonstrator. For this reason, the submission is **not anonymous**: write your name **on each page**.

In your report, reproduce all computer output using a typewriter font. (Use the font `Courier New` if you use Microsoft WORD, and use the `verbatim` environment if you use LaTeX.) The report should also include, as an appendix, your sourcecode in typewriter font. The source code should be presented pleasingly, with self–documenting names for the predicates; you should use comments. The main body of the report should describe each predicate you have used and the reasons for defining them the way you have. For full marks, not only the software should behave as specified but the report should also be pleasing.

Finally, illustrate your answers with **examples different from mine**. (THIS REQUIREMENT IS ESSENTIAL!) Furthermore, you must follow the *Essential Requirements* given below.

Essential Requirements:

You will loose marks if you do not follow these specifications!

- **Hand out week:** In Semester 2, Week starting 06/02/2017 (and ending 10/02/2017): An electronic copy of this coursework sheet (a pdf file) will be available on the module Blackboard site by the beginning of this week.
- **Submission deadline:** 6th March 2017 (Monday), by 4.00 pm.
- **Submission as prescribed by the University Rules.**
- **Items to be submitted:** Submit your full report and as part of it also submit your source code as specified above.
- **You must write on the first page of your submission your name and student number.**

Other requirements:

- The Linux version of SWI-Prolog is available and you are recommended to use this version to cut-and-paste your interactive session into your document.
- You should invoke SWI-Prolog from the command line terminal by simply typing *swipl*.
- You should use one single Prolog source file in your working.
- Finally, as I have repeatedly said in the lectures and is written in the module descriptor, this coursework is 12.5% maximum of the total assessment percentage achievable for this module.

Modelling Queue Manipulations

Assumptions and Starting Point

Assume that we have the plain text file called `Cw3_16-17.pl` comprising the following Prolog code,

```
:- dynamic queue/1.

queue([adam,eve,tracy,pete,john,james,ruth,david]).
```

This Prolog code is modelling a queue of people at a bank counter; IN OUR EXAMPLE initially, Adam is the head of the queue, David is the last in the queue.

The predicates you are asked to define are intended to manipulate this queue **AND ALL OTHER SIMILAR QUEUES**.

Notice. The examples below I have generated consecutively starting with the 8 people queue as shown above. As I progress, the queue of course evolves accordingly.

Your Tasks

1. Write a predicate *queue_length/1* for finding the length of the queue. Example:

```
?- queue_length(L).
L = 8.
```

2. Write an *interactive query* for removing the head of the queue. Example:

```
?- ...
?- queue(Q).
Q = [eve, tracy, pete, john, james, ruth, david].
```

   (The query I am looking for is indicated by ....)

3. Define a predicate *remove_head/0*. Example:

```
?- remove_head.
true.
?- queue(Q).
Q = [tracy, pete, john, james, ruth, david].
```

4. Define a predicate *remove_last/0*. Example:

```
?- remove_last.
true.
?- queue(Q).
Q = [tracy, pete, john, james, ruth].
```

5. Define a predicate *join_front/1*. Example:

```
?- join_front(melanie).
true.
?- queue(Q).
Q = [melanie, tracy, pete, john, james, ruth].
```

6. Define a predicate *join_as_last/1*. Example:

```
?- join_as_last(zoe).
true.
?- queue(Q).
Q = [melanie, tracy, pete, john, james, ruth, zoe].
```

   You should consider using the built in predicate *append/3* as it will make your definition very concise.

7. Define a predicate *send_to_back/0* which makes the first person in queue to be become the last one. Example:

```
?- send_to_back.
true.
?- queue(Q).
Q = [tracy, pete, john, james, ruth, zoe, melanie].
```

   You should consider using the built in predicate *append/3* as it will make your definition very concise.

8. Define a predicate *every_second_out/0* which makes the first person to remain, the secon person to leave, the third person to stay, the fourth person to leave etc. Example:

```
?- every_second_out.
true .
?- queue(Q).
Q = [tracy, john, ruth, melanie].
```

   Use the accumulator technique with three accumulator arguments in the auxiliary predicate. (This is a more ambitious task.)

9. Define a predicate *behind/2* for finding out who is behind the person named in the first argument. Example:

```
?- behind(john, B).
B = ruth .
```

   You should use the built in predicate *append/3* for a concise definition.

10. Define a predicate *jump_to_front/1* for making a named person to jump the queue. Example:

```
?- jump_to_front(ruth).
true .
?- queue(Q).
Q = [ruth, tracy, john, melanie].
```

    You should use the built in predicate *append/3* for a concise definition.