CPS721: Assignment 4

Due: November 14, 2023, 9pm
Total Marks: 100 (worth 4% of course mark)
You MUST work in groups of 2 or 3

Late Policy: The penalty for submitting even one minute late is 10%. Assignments are not accepted more than 24 hours late.

Clarifications and Questions: Please use the discussion forum on the D2L site to ask questions as they come up. These will be monitored regularly. Clarifications will be made there as needed. A Frequently Asked Questions Page will also be created. You may also email your questions to your instructor. Check the D2L forum and frequently asked questions first.

Collaboration Policy: You can only discuss this assignment with your group partners or with your CPS721 instructor. By submitting this assignment, you acknowledge that you have read and understood the course policy on collaboration as stated in the CPS721 course management form.

PROLOG Instructions: When you write your rules in PROLOG, you are not allowed to use ";" (disjunction), "!" (cut), and "->" (if-then). You are only allowed to use ";" to get additional responses when interacting with PROLOG from the command line. Note that this is equivalent to using the "More" button in the ECLiPSe GUI.

We will be using ECLiPSE Prolog release 6 to mark the assignments. If you run any other version of PROLOG, it is your responsibility to check that it also runs in ECLiPSE Prolog release 6.

SUBMISSION INSTRUCTIONS: You should submit ONE zip file called `assignment4.zip` containing 3 files:

```
music_site.pl
test_queries.txt
interaction.txt
```

The latter 5 files have been given to you and you should fill them out using the format described. Your submission should not include any other files. If you submit a `.rar`, `.tar`, `.7zip`, or other compression format aside from `.zip`, you will lose marks. All submissions should be made on D2L. Submissions by email will not be accepted. As long as you submit your assignment with the file name `assignment2.zip` your group will be able to submit multiple times as it will overwrite an earlier submission. You do not have to inform anyone if you do. The time stamp of the last submission will be used to determine the submission time. Do not submit multiple `zip` files with different names. If you do, we will use the last submitted one, but you may lose marks.

If you write your code on a Windows machine, make sure the files are saved on plain text and are readable on Linux machines. Ensure your PROLOG code does not contain any extra binary symbols and that they can be compiled by ECLiPSE Prolog release 6.

1 Building a Database [10 marks]

This assignment will exercise what you have learned about natural language understanding and processing. To do so, you will be developing a "smart" website for answering questions about a music database. Ultimately, we would like to be able to interact with the system with statements like "what is the album released in 2022 by Taylor Swift." For this assignment, we will restrict the system to only handle noun phrases, just as we did in class.

To build such a system, you will need a database of facts, a lexicon, and a parser. In this question, you will build the database. To do so, add atomic propositions to the file `music_site.pl` in the section called `database`. The atomic propositions should ONLY use the following predicates:

- `albumArtist(AlbumName, ArtistName)` - an artist of the album. May be a person or a band.
- `albumYear(AlbumName,Year)` - the year the album was released.
- `albumGenre(AlbumName, Genre)` - the genre of the album (*ie.* `rock`, `jazz`, `pop`)
- `trackList(AlbumName, List)` - the track list for the given album. List is a standard Prolog list of `song(SongName, SongLength)` terms, where `SongName` is the name of the song, and `SongLength` is the length of the song in seconds (*ie.* `190` means the song is 190 seconds long, or 3 minutes and 10 seconds).

An example of a track list for an album would be as follows (only 3 songs are listed here for space reasons):

```
trackList(views,
          [song(keep_the_family_close, 328), song(9, 255), song(u_with_me, 297)]).
```

The first song in the list should be undersood as the first song on the album, the second song in the list as the second song on the album, etc.

Your knowledge base should contain at least 10 albums and at least 20 total songs. In addition, consider the following:

- You will use this knowledge base for testing, so we suggest you read the rest of the assignment first to see what kind of statements will be useful to fully test your system.
- As on Assignment 1, use constants for your artist, album, and song names. For simplicity, you should also not include any punctuation or accents in names as well.
- You can assume that every artist, album name, and song name is unique. That is, no two artists have an album with the same name and no two albums have a song with the same name.
- Every album has exactly one artist who released it, exactly one release year, and exactly one track list.
- An album may have more than one genre.
- For this assignment, we assume albums have no more than 20 songs on them.

2 Helpers [15 marks]

For this question, you will write predicates that will be useful for defining your lexicon and adding other language features to your system. This will also give you more experience with recursion on lists and creating rules for interacting with a knowledge base with terms in it. All of these helpers, and any others you decide to include in your system should be put in section `helpers` in `music_site.pl`.

a. [**2 marks**] Create a predicate `isSong(Name)` which takes in a name, and checks if there is a song on some album with that name. For example, if the knowledge base contains only the album `views` and the track list in Question 1, `isSong(u_with_me)` will be true and `isSong(love_me_do)` will be false. If you call `isSong(X)`, you should also be able to use "More" to iterate through all the songs on any album in your database.

b. [**2 marks**] Create a predicate `songLength(Name, Length)` which takes in a song name and finds the length of that song. For example, if the knowledge base contains only the album `views` and the track list in Question 1, then `songLength(keep_the_family_close, X)` should succeed with `X = 328`. You should also be able to use this predicate as `songLength(X, 200)` to find a song that is 200 seconds long or as `songLength(X, Y)` to iterate through all songs and their lengths.

c. [**2 marks**] Create a predicate `onAlbum(SongName, AlbumName)` which takes in a song name and finds the album that song is on. For example, if the knowledge base contains only the album `views` and the track list in Question 1, then `onAlbum(keep_the_family_close, X)` should succeed with `X = views`. You should also be able to use this predicate as `onAlbum(X, views)` to iterate through the songs on `views`, or `onAlbum(X, Y)` to iterate through all song name-album pairs.

d. [**4 marks**] Create a predicate called `albumLength(AlbumName, Length)` which calculates the length of an album. The length of an album is given by the sum of the lengths of all the songs on that album. For example, if the knowledge base contains only the album `views` and the track list in Question 1, then `albumLength(views, X)` should succeed with `X = 880`. You should also be able to use `albumLength(X, 880)` to find an album of length `880` seconds and `albumLength(X, Y)` to iterate through all album name-length pairs.

e. [**5 marks**] For this question you will create a helper for answering questions like "What is the second song on views?" In particular, create a predicate called `atNamedIndex(List, IndexName, Element)` which takes in an arbitrary Prolog list, an index name (*ie.* `fourth`), and finds the element at the index corresponding to that name. For example, the following should hold:

- `atNamedIndex([a, b, c, d, e], third, c).`
- `atNamedIndex([1, 2, 3, 4, 5, 6], second, X)` with `X = 2`.
- `atNamedIndex([a, b, c, d, e], X, e)` with `X = fifth`.

You can assume that while either of `IndexName` or `Element` (or both) may be set to a variable, the input `List` will always be given.

You can use the `orderNames` predicate to get the list of the first 20 such names when writing your program. This means that you also only have to handle the numbers from `first` to `twentieth`.

3 Building a Lexicon [55 marks]

You will now build a lexicon, as we did in class, of articles, common nouns, proper nouns, adjectives, and prepositions. Each should be added to the corresponding section in `music_site.pl`. Your lexicon should allow for queries of the following type:

- `what([a, rock, song, by, the_beatles], S).`
- `what([the, genre, of, hey_jude], G).`
- `what([a, new, album, by, a, pop, artist], A).`
- `what([the, length, of, views], L).`
- `what([the, first, song, on, views], S).`
- `what([an, old, album, by, the_mountain_goats], A).`
- `what([the, release_year, of, hey_jude], Y).`
- `what([a, short, beyonce, song, released_in, 2012], S).`
- `what([the, release_year, of, the, short, drake, album], Y).`
- `what([a, record, released_in, 2005, with, a, long, song], A).`
- `what([a, drake, album, released_before, views], S).`

Below, we describe what your lexicon must include. Feel free to add additional words if you would like to extend the functionality of your system, though you will not get marks for doing so.

Article

Your lexicon should include the articles `a`, `an`, `the`, and `any`. For the purpose of this assignment, the meaning of these is all equivalent. For example, both `[an, album, ...]` and `[a, album, ...]` are valid.

Common Nouns

Your lexicon should include the common nouns `album`, `record`, `artist`, `song`, `track`, `length`, `genre`, and `release_year`. Note the following

- For this assignment, `album` and `record` are equivalent and can be used interchangeably. The same is true of `song` and `track`.
- The common noun `length` may refer to either the length of a song or the length of an album.

HINT: Once you have added some articles and common nouns, you can start testing your system with queries such as `what([a, album], X)`, which should allow you to iterate over all the albums by calling "More".

Proper Nouns

Your lexicon should include all album names, artist names, and song names as proper nouns. We will be testing your system using our own knowledge base, so your rules should be able to handle having the knowledge base changed to have other names. Any number (like year or song length) is also considered as proper nouns. You may use the library predicate `number(X)`, which is true if `X` is a number, for this purpose.

Note that proper nouns are a referent to themselves and thus, you can handle them with rules of the following form:

```
proper_noun(X) :- person(X).    % X is the name of the person in this context
```

Adjectives

Your lexicon should define the following adjectives:

- `short` - a song is "short" if it is less than 3 minutes long. An album is "short" if it is less than 10 minutes long.
- `long` - a song is "long" if it is at least 6 minutes long. An album is "long" if it at least an hour long.
- `old` - an album or song is "old" if it was released before the year 2000.
- `new` - an album or song is "new" if it was released in the current year. Note that the current year is defined using the predicate `currentYear`. We have provided an example in the `constants` section of `music_site.pl`, however your rules should work if we change the year.
- Each genre can be used as an adjective. In particular, you should be able to refer to `[a, pop, album]`, a `[a, pop, artist]`, and `[a, pop, song]`, though for any arbitrary genre that appears in the knowledge base. An artist is in a given genre if they have at least one album in that genre. A song is in a given genre if it is on an album of that genre.
- Each artist can be used as an adjective. This is to allow for queries like `[a, taylor_swift, album]` or a `[a, taylor_swift, song]`.
- `first`, `second`, ... - a song is the **n**-th song on an album if it is the **n**-th element in the track list of that album.

HINT: Once you have added some articles, common nouns, and adjectives, you can start testing your system with queries such as `what([a, rock, song], X)`, which should allow you to iterate over all the rock albums by calling "More".

Prepositions

Your lexicon should define the following adjectives:

- `on` - to refer to a song **on** an album .
- `by` - to refer to a song or album **by** an artist.
- `released_in` - to refer to the year that a song or album was **released in**.
- `of` - can be used to refer to the length **of** a song or album, the release year **of** a song or album, the genre **of** a song or album.
- `released_before` and `released_after` - can be used to refer to an album that was released before (or after) another album. You do not have to handle phrases of the form "an album released before 2012" for this part (it will be considered in question 5). Note that you only need to handle comparisons between two albums, not between two songs or an album and a song. You may additionally add this functionality, but we will not give marks for it.

4 Testing the Parser [10 marks]

For this question, you will test your lexicon by trying queries using the `what` predicate and the simple noun phrase parser given in class. This parser has already been added to `music_site.pl`.

Test the `what` predicate using queries like those given above, showing that your system is capable of identifying the entities being referred to by your noun phrases. You should test **at least 10 new phrases** in addition to those given at the beginning of Question 3. The new phrases should be selected to help convince the TA that your system is correct. Put your queries and a brief description (in plain English of what each is trying to test), in the file called `q4_tests.txt`. Do not include the queries given to you Question 3 in this document. If you complete all or part of Question 5, you can include your tests from that part in this file. You will lose marks if you do not have at least 10 new queries.

In addition, include the interaction you had with Prolog when using those queries Included in Question 3 AND your new queries in the file `q4_interaction.txt`.

In addition, please note the following:

- This part of your submission will not be evaluated on whether your systems succeeds or not on your queries, but on the correctness of your queries. As such, we may use your knowledge base and queries on our system to evaluate if they are correct.
- As part of your interaction, you should test the output of your program when you ask for more solutions. It is fine, and generally expected, if you get duplicate answers. However, you should ensure that you only get correct answers. You will lose marks if your program produces any incorrect answers or misses any correct answers.

Note, this part of your submission will not be evaluated on whether your systems succeeds or not on your queries, but on the correctness of your queries. As such, we may use your knowledge base and queries on our system to evaluate if they are correct.

5 Additional Features [20 marks]

DO NOT ATTEMPT THIS PART UNTIL ALL PREVIOUS PARTS HAVE BEEN COMPLETED.

In this part of the assignment, you will be adding additional English features to your system. You may use the built-in predicate `number(X)` for this part. This predicate is true if the given value is a number. Otherwise it is false.

a. [**5 marks**] Add the preposition "with", along with additional rules for "of", that allow you do queries of the following form:

- `what([a, taylor_swift, song, with, a, length, of, 200], S).`
- `what([a, country, album, with, a, release_year, of, 2013], Y).`

You should be able to use these prepositions to refer to a song or album with a particular release year, or a song or album with a particular length. You do not need handle genre in this way (*ie.* phrases of the form "with the genre of jazz").

The rules needed for this part should be added to `preposition` part of `music_site.pl`.

b. [**5 marks**] Improve the functionality of `released_before` and `released_after` to allow for queries of the following form:

- `what([a, taylor_swift, album, released_before, 2017], A).`
- `what([a, taylor_swift, album, released_before, 2017, with, a, length, of, 200], A).`

In particular, your program should handle any arbitrary input year, not just years for which there is an album in your database. Your program should also be able to do so without introducing a `year` predicate like in assignment 1. As before, we will only test this functionality for albums, and not for songs.

HINT: Update the `parser` section of `music_site.pl` to add this functionality.

c. [**5 marks**] Add to your system the ability to handle prepositional phrases of the form "between X and Y". For example, your system should be able to handle the following:

- `what([an, album, with, a, release_year, between, 2020, and, 2022], A).`
- `what([an, song, with, a, length, between, the, length, of, hey_jude, and, 600], S).`

HINT: Update the `parser` section of `music_site.pl` to add this functionality.

d. [**5 marks**] Add to your system the adjective `oldest` which can be used as follows:

- `what([the, oldest, album], A).`
- `what([the, oldest, rock, album, by, the_beatles], A).`

We will only test this functionality for albums, and not for songs. You may also add that if you so choose, but you will not get more marks. In addition, note that when there is a sequence of adjectives, `oldest` must be the first one. That is, we can have phrases like `[the, oldest, short, taylor_swift, song]` but we cannot have phrases of the form `[the, short, oldest, taylor_swift, song]`.

HINT: Update the `parser` section of `music_site.pl` to add this functionality.