CSCI.4430 Programming Languages Fall 2026
Programming Assignment #3
Due Date: 7:00 PM Nov 30th

In this assignment, you will use **SWI-Prolog** to implement a simple music-request system and a playlist-generation system. Both parts use the same song knowledge base, and may be implemented and graded independently. **Modify only `main.pl`.**

Song Knowledge Base

The provided *songs.pl* file represents one table using facts of the form

```
song(Name, Type, Artist, Duration).
```

Here `Name`, `Type`, and `Artist` are lowercase Prolog atoms. `Duration` is a whole number of minutes. Song names are unique; multiword names use underscores. Below is the complete **10-song example dataset** in *songs.pl*:

Name	Type	Artist	Duration
rocket_man	rock	elton_john	5
shape_of_you	pop	ed_sheeran	4
perfect	pop	ed_sheeran	4
happier_than_ever	rock	billie_eilish	5
bad_guy	pop	billie_eilish	3
skyfall	pop	adele	5
rolling_in_the_deep	soul	adele	4
good_4_u	rock	olivia_rodrigo	3
drivers_license	pop	olivia_rodrigo	4
levitating	dance	dua_lipa	3

The types and whole-minute durations are **assignment data**, rather than official genre labels or exact recording lengths. For example:

```
song(rocket_man, rock, elton_john, 5).
song(shape_of_you, pop, ed_sheeran, 4).
```

The 10 rows are for examples and public testing only. The grader may use additional or replacement song facts. **Do not hardcode song names or expected answers.**

1 Music Request Processing (45 points)

In Part 1, you will parse short English commands about songs and use the song database to produce matching titles. The supported language is deliberately limited to the **eight request patterns** below.

Grammar

Filter	All songs	One song
Artist	`find songs by ARTIST`	`find one song by ARTIST`
Type	`find songs of type TYPE`	`find one song of type TYPE`
Duration	`find songs shorter than N minutes`	`find one song shorter than N minutes`
Type + duration	`find songs of type TYPE shorter than N minutes`	`find one song of type TYPE shorter than N minutes`

The equivalent grammar is:

```
<request> ::= "find" <subject> <filter>
<subject> ::= "songs" | "one song"
<filter>  ::= "by" ARTIST
            | "of type" TYPE
            | "shorter than" N "minutes"
            | "of type" TYPE "shorter than" N "minutes"
```

`ARTIST` and `TYPE` are single lowercase Prolog atoms (underscores are allowed). `N` is a nonnegative integer; the duration comparison `Duration < N` is strict. Requests are case-insensitive, have no punctuation, and consist of whitespace-separated words. The provided *helper.pl* converts text lines to lowercase atom tokens; for example, the number `4` becomes the atom `'4'` before parsing. The grammar remains valid when the artist or type is not present in the song database.

Objective

Implement **one required predicate** in *main.pl*: `song_query(Tokens, Songs).` (**45 points**)

Given a list of word tokens, interpret the request and return matching song names from the database:

1. **Parsing (25 points):** Recognize the eight supported query patterns, including single-song and multiple-song requests.
2. **Song lookup and output (20 points):** Use the supplied song facts, not hardcoded answers. For `songs`, return all distinct matching song names, sorted alphabetically (by Prolog atom order). For `one song`, return the alphabetically first match as a *one-element list*.

A **valid request with no matches** succeeds with `Songs = []`. An **invalid request** (missing or extra tokens, or an unsupported pattern) **fails**. You may choose any internal representation and helper predicates; there is **no required standalone parser predicate**. The only Part 1 interface tested is `song_query(Tokens, Songs)`.

The provided *helper.pl* tokenizes input lines, calls `song_query(Tokens, Songs)`, and prints the results. You do not need to handle file input/output.

Expected Output

The following examples use the 10-song public database. Every call goes directly to the **single required predicate**.

```
?- song_query([find,songs,by,adele], Songs).
Songs = [rolling_in_the_deep,skyfall].

?- song_query([find,songs,of,type,rock], Songs).
Songs = [good_4_u,happier_than_ever,rocket_man].

?- song_query([find,one,song,by,ed_sheeran], Songs).
Songs = [perfect].

?- song_query([find,songs,shorter,than,'4',minutes], Songs).
Songs = [bad_guy,good_4_u,levitating].

?- song_query([find,songs,of,type,pop,
              shorter,than,'4',minutes], Songs).
Songs = [bad_guy].

?- song_query([find,songs,of,type,jazz], Songs).
Songs = [].

?- song_query([find,song,by,adele], Songs).
false.

?- song_query([find,songs,by,adele,please], Songs).
false.
```

Part 1 input-file format. The file *example_part1.txt* contains one English request per line. The helper prints one sorted answer list per valid request, or `false` when the grammar is invalid. Blank lines and lines beginning with `#` are ignored.

For example, *example_part1.txt* may contain:

```
# Example Part 1 requests
find songs by ed_sheeran
find one song of type rock

find songs shorter than 4 minutes
find songs of type pop shorter than 4 minutes
find song by adele
```

2 Playlist Generation (45 points)

In Part 2, you will generate playlists that satisfy user-provided limits on duration and number of songs. The limits are **input parameters**, not fixed constraints. For example, `make_playlist(30,45,10,P)` requests a playlist between 30 and 45 minutes, with at most 10 songs. Public tests use smaller ranges to keep expected outputs short.

Rules

Every generated playlist must satisfy all five rules:

1. Every song comes from the supplied *songs.pl* facts.
2. Total duration is between `MinTime` and `MaxTime`, inclusive.
3. The playlist contains at least one song and at most `MaxSongs` songs.
4. Songs cannot be repeated.
5. Song names appear in alphabetical (Prolog atom) order. Generate a given song combination *only once*; different permutations are not different playlists.

You may assume the three numeric inputs are integers satisfying `0` $\leq$ `MinTime < MaxTime` and `MaxSongs` $\geq$ `1`. The order of *different playlists* during backtracking is not prescribed. You must enumerate **all** valid playlists without duplicate solutions.

Objective

Complete the following two predicates in *main.pl*:

1. `total_duration(Playlist, Minutes)` (**15 points**). Compute the combined length of all titles in the list using the supplied song facts. The empty list has duration `0`. An unknown title causes failure. Do not hardcode song durations.
2. `make_playlist(MinTime, MaxTime, MaxSongs, Playlist)` (**30 points**). Generate one valid, alphabetically ordered playlist per solution. Further backtracking must enumerate all remaining valid playlists. If no valid playlist exists, this predicate fails.

Expected Output

These examples use the provided 10-song database. Each request below has only a few solutions.

```
?- total_duration([], T).
T = 0.

?- total_duration([shape_of_you,good_4_u], T).
T = 7.

?- make_playlist(15, 16, 3, P).
P = [happier_than_ever,rocket_man,skyfall].

?- make_playlist(10, 11, 2, P).
P = [happier_than_ever,rocket_man] ;
P = [happier_than_ever,skyfall] ;
P = [rocket_man,skyfall].
```

```
?- make_playlist(11, 12, 2, P).
false.
```

The `make_playlist` predicate may return valid playlists in a different order; however, the song names *within* each playlist must be alphabetical. The 10–11 minute example has exactly three solutions (pairs of distinct 5-minute songs). The 15–16 minute example has exactly one solution with at most three songs.

Part 2 input-file format. Each test in *example_part2.txt* is a Prolog term ending with a period. The helper collects and sorts complete lists of generated playlists for each request:

```
test(total_duration([])).
test(total_duration([shape_of_you,good_4_u])).
test(make_playlist(15,16,3)).
test(make_playlist(10,11,2)).
test(make_playlist(11,12,2)).
```

The expected file output is:

```
total_duration([]) => 0
total_duration([shape_of_you,good_4_u]) => 7
make_playlist(15,16,3) =>
  [[happier_than_ever,rocket_man,skyfall]]
make_playlist(10,11,2) =>
  [[happier_than_ever,rocket_man],
   [happier_than_ever,skyfall],
   [rocket_man,skyfall]]
make_playlist(11,12,2) => []
```

A printed `[]` means there are no valid playlists. Public tests intentionally yield just a few answers. Hidden tests may use an expanded song database or other song facts; do not assume a fixed number of songs.

Provided Files

- *main.pl*: The only file to modify. Three required predicate stubs: one for Part 1 and two for Part 2. You may define additional helpers in this file.
- *songs.pl*: The 10-song example table. Do not modify. Hidden grading may use a larger or replacement song database.
- *helper.pl*: Provided line tokenizer, file reader, command-line dispatcher, and result formatter. Do not modify.
- *example_part1.txt* and *expected_part1.txt*: Public English request examples and matching output.
- *example_part2.txt* and *expected_part2.txt*: Public playlist test cases and matching output.

How to Run in SWI-Prolog

Run these commands from the directory containing the provided files:

```
swipl -s main.pl -t main --quiet -- example_part1.txt part1 > result_1.txt
swipl -s main.pl -t main --quiet -- example_part2.txt part2 > result_2.txt
```

The arguments after `--` specify the input file and part. The expected files show output from a completed solution. Compare your output with the supplied expected results.

Submission and Grading

Total: **100 points**. Part 1 is **45** (25 parsing + 20 song lookup/output), Part 2 is **45** (15 duration + 30 playlist generation), and Testing/Code Quality is **10**.

Submit `main.pl` and a short `readme.txt` with names, implementation notes, testing notes, and any known limitations. Do not submit modified copies of the provided data or helper files.

Warning About Posting Solutions Publicly & Inter-Team Code Sharing

Do not post any code you submit for this assignment onto any public website or network share. Also, your code from your team (which can be at most two people) should be wholly yours; Do not look at or use code made by other students and/or teams as your own. If a violation of either of these rules is detected, your grade will be heavily affected, and there may be more serious consequences.

Links & Documentation for (SWI) Prolog

- Quickstart documentation:
  - `https://www.swi-prolog.org/pldoc/man?section=quickstart`
- Installers for version 8.4 (Windows/MAC):
  - `https://www.swi-prolog.org/download/stable` (for Linux, check your package manager for `swi-prolog`)
- Definite Clause Grammar documentation:
  - `https://www.swi-prolog.org/pldoc/man?section=DCG`