Lab 7 Assignment (CSI3120)

Programming in Prolog

Input/Output Operations, DCGs, and Dynamic Data Management

In this lab, students will advance their knowledge of Prolog by implementing dynamic data manipulation, managing runtime data, and exploring Prolog's input/output capabilities. Through practical tasks, students will gain experience in handling files, creating interactive systems, and working with dynamic predicates to modify the knowledge base at runtime. This lab encourages students to enhance their problem-solving skills while using Prolog to develop more interactive and complex applications.

Topics Covered

- **Input and Output in Prolog**: Learn how to read and write terms, characters, and sentences, and perform file operations.
- **Definite Clause Grammars (DCGs)**: Explore how to parse and structure complex patterns using DCGs.
- **Dynamic Predicates**: Implement a library system, where students can add, remove, and manage books at runtime.

The lab consists of three tasks, each focusing on different aspects of input/output handling, parsing, and dynamic data management. Students are encouraged to collaborate in groups of three to complete each task

TASK A: Generating Patterns with User Input

Question 1: Right-Angled Triangle Pattern on Console

Write a Prolog predicate that asks the user for the height of a right-angled triangle and then prints the triangle pattern of that height to the console.
Instructions:

1. Prompt the user to enter an integer representing the height of the triangle.
2. Print a right-angled triangle using #, where each row has one more # than the previous row.

Test Case: If the user enters 4, the output on the console should look like:
?- right_angle_triangle_console.
Enter the height of the right-angled triangle: 4.

#
##

###
####
true .

Question 2: Isosceles Triangle Pattern Written to a File

Generate an isosceles triangle pattern based on a height specified by the user and write it to a file. This pattern resembles the upper part of a diamond.
Instructions:

1. The user specifies an integer as the height.
2. The triangle pattern should be written to the specified file using * symbols.

Test Cases: If the user enters 5 for height and the filename is triangle.txt, the content of the file should look like:
?- isosceles_triangle_pattern_file(5, 'triangle.txt').
Isosceles triangle pattern written to file: triangle.txt
true .

```
    *
   ***
  *****
 *******
*********
```

Report Requirements

1. Objective and Approach
   - Briefly explain the purpose of Task A, highlighting the difference between the right-angled triangle pattern in Question 1 and the isosceles triangle pattern in Question 2.
   - Describe your approach for implementing each pattern, focusing on the recursive structure used in both cases.
2. Console Output vs. File Output
   - Discuss how you handled console output in Question 1 and file output in Question 2.
   - Explain any challenges encountered with file handling in Prolog (if any) and how you addressed them.
3. User Input Handling
   - Describe how you prompted for and handled user input for both height and filename.
   - Discuss any error handling (such as checking that the height is a positive integer) implemented in your code.
4. Recursion and Pattern Generation
   - Explain the recursive logic used to generate each triangle pattern.
   - Describe how the base case and recursive case work together to produce the triangle, and how they differ between the two patterns.
5. Testing and Sample Outputs
   - Provide screenshots or sample outputs for:
     - A right-angled triangle pattern printed to the console.
     - An isosceles triangle pattern written to a file.

Task B: Parsing Game Character Descriptions with Definite Clause Grammars (DCGs)

This task challenges you to use **Definite Clause Grammars (DCGs)** in Prolog to parse structured game data. You'll develop a DCG to represent and validate character descriptions within a computer game, based on specific rules for character types, subtypes, movements, and attributes.

In a fictional computer game, characters are either **enemy** or **hero** characters. Each character type has unique attributes and behaviors that you'll model in Prolog.

Character Attributes

1. **Character Type**: Either enemy or hero.
2. **Character Subtype**:
    - **Enemies**: DarkWizard, Demon, Basilisk
    - **Heroes**: Wizard, Mage, Elf
3. **Sequence**: A positive integer indicating the creation order of the character.
4. **Movement Direction**:
    - **Enemies** always move towards heroes.
    - **Heroes** without weapons (no_weapon) move away from enemies, while those with weapons (has_weapon) move towards.
5. **Health Levels**:
    - Health levels are encoded as very_weak, weak, normal, strong, and very_strong.
6. **Weapon**: Only heroes can have weapons (has_weapon or no_weapon).
7. **Movement Style**: Characters can move jerky, stealthy, or smoothly.

Each character description follows this format:
CharacterType,CharacterSubType,Sequence,MovementDirection,Health,Weapon,MovementStyle

Instructions

1. **Define a DCG for Character Descriptions**
    - Write a DCG to parse each part of the description.
    - Ensure that:
        - **Enemies** always move towards.
        - **Heroes** move away if they have no_weapon and towards if they have weapon.
        - Sequence is a positive integer.
        - **Heroes** can have either has_weapon or no_weapon, while **enemies** always have no_weapon.
2. **Write the DCG Rules**
    - Implement rules for:
        - **Character type** and **subtype**
        - **Sequence** number
        - **Movement direction** based on character type and weapon possession
        - **Health levels**
        - **Weapon possession** for heroes and enemies
        - **Movement styles**
    - If a description does not match the required format or violates a rule, the DCG should fail.

Test Cases:

?- phrase(character_description, [enemy, demon, 7, towards, strong, no_weapon, stealthy]).
% Expected Output: true
?- phrase(character_description, [hero, wizard, 12, towards, normal, has_weapon, smoothly]).
% Expected Output: true
?- phrase(character_description, [hero, elf, 5, towards, very_strong, no_weapon, jerky]).
% Expected Output: false

Report Requirements

In addition to submitting your code, provide a brief report addressing:

1. **DCG Explanation**: Describe each DCG rule, explaining how they ensure valid descriptions.
2. **Testing and Results**: Present results of various valid and invalid queries, explaining why they pass or fail.
3. **Challenges**: Discuss any challenges and insights you gained while working with DCGs.

Task C: Library Management System with Prolog

Design and implement a Library Management System in Prolog using **dynamic predicates**. The system should allow users to add, remove, borrow, return, search, and recommend books. This task will enhance your understanding of **dynamic knowledge base manipulation**, **runtime data management**, and **querying** in Prolog.

Requirements

1. **Define Book Predicate**
    - Create a dynamic predicate book/4 to represent each book in the library.
    - Each book should have the following attributes:
        - **Title**: The title of the book.
        - **Author**: The author of the book.
        - **Year**: The publication year of the book.
        - **Genre**: The genre of the book.
    - Example: book('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Novel').
2. **Add Book**
    - Write a predicate add_book/4 to add a book to the library. Use assertz predicate.
    - Ensure that duplicate entries are not allowed.
3. **Remove Book**
    - Write a predicate remove_book/4 to remove a specific book from the library. Use retract predicate.
4. **Check Availability**
    - Write a predicate is_available/4 to check if a specific book is available in the library.
5. **Borrow Book**
    - Write a predicate borrow_book/4 to allow borrowing of a book if it exists and is not currently borrowed.
    - Maintain a dynamic predicate borrowed/4 to track the borrowed status of each book.
6. **Return Book**
    - Write a predicate return_book/4 to mark a borrowed book as returned, making it available

again in the library.

7. **Search Options**
    - Implement predicates to find books by:
        - **Author** (find_by_author/2): Retrieve all books by a specific author.
        - **Genre** (find_by_genre/2): Retrieve all books of a particular genre.
        - **Year** (find_by_year/2): Retrieve all books published in a specific year.
8. **Book Recommendations**
    - Write two recommendation predicates:
        - **Recommend by Genre** (recommend_by_genre/2): Suggest books of a specified genre.
        - **Recommend by Author** (recommend_by_author/2): Suggest books by the same author.

Test Cases:

% Adding initial books to the library.
?- add_book('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Novel').
true.
?- add_book('1984', 'George Orwell', 1949, 'Dystopian').
true.
?- add_book('To Kill a Mockingbird', 'Harper Lee', 1960, 'Novel').
true.
?- add_book('Brave New World', 'Aldous Huxley', 1932, 'Dystopian').
true.

1. Adding a Book
?- add_book('Reminders of Him', 'Colleen Hoover', 2022, 'Novel').
true.

2. Remove a Book
?- remove_book('1984', 'George Orwell', 1949, 'Dystopian').
true.

3. Check Availability
?- is_available('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Novel').
true.

?- is_available('1984', 'George Orwell', 1949, 'Dystopian').
false.

4. Borrowing a Book
?- borrow_book('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Novel').
true.
?- is_available('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Novel').
false.

5. Return a Book
?- return_book('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Novel').

true.
?- is_available('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Novel').
true.

6. Find Books
?- find_by_genre('Novel', Books).
Books = ['The Great Gatsby', 'To Kill a Mockingbird', 'Reminders of Him'].
?- find_by_year(1925, Books).
Books = ['The Great Gatsby'].
?- find_by_author('Harper Lee', Books).
Books = ['To Kill a Mockingbird'].

7. Recommendation
?- recommend_by_genre('Novel', Recommendations).
Recommendations = ['The Great Gatsby', 'To Kill a Mockingbird', 'Reminders of Him'].

Report Requirements:

In addition to submitting your code, please prepare a brief report covering the following aspects:

1. **Explanation of Dynamic Predicates**: Describe the purpose and function of dynamic predicates in your program.
2. **Explanation of Each Predicate**: Provide a brief explanation of how each predicate (add, remove, borrow, return, etc.) operates within the library system.
3. **Testing Evidence**: Include sample queries and expected outputs to validate each functionality.

Lab Submission Guidelines

The due date for each lab is two weeks from the date of the lab session. For Lab 6, the submission deadline is **November 21st, 11.59 PM**. Each group is required to make a single submission on Brightspace.

1. One code file for each task in prolog file (Eg: filename.pl).
2. One PDF report explains your work concisely and includes a few screenshots of your output against the test case provided.
3. Zip everything into one folder and please upload it to Brightspace.

On the front page of your report, please include:

- Group number
- Names of all group members
- Student IDs

You will be evaluated as a group unless one of the members raises concerns about unequal contributions. In such cases, we may ask for further clarification and reserve the right to evaluate members individually.

If you use or receive help from ChatGPT or similar tools, you must clearly reference it in your submission. Failure to do so will result in a **zero** for the lab, and further actions may be taken in accordance with university policies.

You are allowed to use tools like ChatGPT for assistance, but **copying and pasting** the generated results directly into your assignment will not be tolerated. You are expected to **understand the help provided and implement the solution in your own words and approach**.