Lab 8 Assignment (CSI3120)

Programming in Prolog

Prolog Applications

In this lab, students will apply Prolog programming techniques to create two applications: A Travel Planner and a simplified Minesweeper Game.

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

TASK A: Travel Planner

In this task, you will build a Travel Planner using Prolog to manage a journey. You will apply dynamic predicates, recursion, backtracking, file I/O, and Definite Clause Grammars (DCGs) to develop a system that allows users to add destinations, track expenses, set budgets, and filter destinations based on specific criteria.

Requirements:

1. Destination Management
    - Implement a dynamic predicate destination/4 to store travel destinations with the following attributes:
        - Name: Destination name (e.g., "Paris").
        - Start Date: Date of arrival.
        - End Date: Date of departure.
        - Budget: Budget allocated for this destination.
    - Write predicates to dynamically add and remove destinations.
2. Expense Tracking
    - Define a dynamic predicate expense/3 to track expenses associated with each destination:
        - Destination: Name of the destination (e.g., "Paris").
        - Category: Expense category (e.g., "food", "transport").
        - Amount: Cost of the expense.
    - Implement a recursive predicate to calculate total expenses for each destination.
3. Budget Validation
    - Write a predicate to set and validate a budget for each destination. Ensure the

predicate checks whether total expenses exceed the budget.

4. Filtering Destinations and Expenses
    - Write predicates to filter destinations by date or expense category. Implement backtracking control with the cut operator where necessary.
5. Command Parsing with DCGs
    - Use Definite Clause Grammars (DCGs) to parse user commands, including:
        - Adding a destination
        - Removing a destination
        - Listing expenses for a destination
        - Checking budget
6. Saving and Loading Journey (File I/O)
    - Implement predicates to save and load the journey from a file, ensuring the file includes both destinations and expenses. Include error handling for cases like missing files or incorrectly formatted data.

Report Requirements

In addition to submitting your code, include a brief report covering the following:

1. **Dynamic Predicates**: Explain how you used assert and retract to dynamically manage destinations and expenses.
2. **Recursion and Backtracking**: Describe how you handled recursive calculations for expenses and used the cut operator to control backtracking.
3. **DCG Implementation**: Provide examples of DCG rules you used to parse commands, with an explanation of each rule's purpose.
4. **File I/O and Error Handling**: Describe how you implemented saving and loading functionality, with attention to error handling.
5. **Testing and Results**: Present results of running key test cases. Include screenshots or console output demonstrating successful execution of each feature.

TASK B: Minesweeper Game

In this task, you will create a simplified version of the classic Minesweeper game on a 6x6 grid using Prolog. The grid will contain hidden mines that the player must avoid while uncovering safe cells. You will use Prolog's features to handle game logic, display, and interaction.

Requirements

1. Game Setup
    - **Grid Initialization**: Define a 6x6 grid where each cell has a random chance of containing a mine. Place a total of 6 mines across the grid, ensuring no overlapping.
    - **Dynamic Predicates**: Use dynamic predicates to store the locations of mines, as well as to track cells that have been revealed by the player.
2. Cell Checking and Neighbour Count

   - **Revealing Cells**: Create a predicate reveal_cell/2 that takes a row and column as input and reveals the contents of the cell (either a mine or a count of adjacent mines).
   - **Neighbouring Mines Count**: Implement logic to count and display the number of mines surrounding each cell, which is revealed when a player checks a safe cell.

3. Game Interaction
   - **User Commands**: Allow players to interact with the game by:
     - Checking a cell to reveal its contents.
     - Flagging a cell as containing a mine.
     - Unflagging a cell.
   - **End of Game Conditions**: The game ends when either:
     - The player successfully flags all mines without any errors.
     - The player reveals a mine, resulting in a loss.

4. Display and Feedback
   - **Game Board Display**: Write predicates to display the game board, indicating flagged cells, revealed mine counts, and hidden cells.
   - **Error Handling**: Include feedback for invalid moves, such as attempting to reveal an already revealed cell or exceeding grid boundaries.

Report Requirements

In addition to submitting your code, include a brief report covering the following:

1. **Game Logic and Dynamic Predicates**: Explain how you set up the 6x6 grid, assigned mines, and used dynamic predicates to track game state.
2. **Recursive Logic and Neighbouring Count Calculation**: Describe how you implemented recursive checks to calculate adjacent mines and handled player moves.
3. **Game Interaction and Display**: Explain how you managed user commands and displayed the game board for each move.
4. **Testing and Results**: Provide test cases and the results of playing through various game scenarios. Include screenshots or console outputs to demonstrate key game interactions.

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 28th, 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.