Lab 7: Input/Output Operations, DCGs, and Dynamic Data Management

CSI 3120 A - Programming Language Concepts

Fall 2024

School of Electrical Engineering and Computer Science
University of Ottawa

Anthony Le - 300287511
Yaroslav Volkov - 300235591

Experiment Date: Nov 8, 2024
Submission Date: Nov 21, 2024

Lab tasks and their desired outputs

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.

Task A:

1. Objective and Approach

In order to properly print out the triangles for Task A, we decided to utilize a recursive approach in order to properly show the expected outputs, by printing the necessary lines with the number of hashtags or asterisks to give the proper structure of the triangle. Most notably, the two sub-tasks that we were required to implement, had certain differences between them as we had to also account for the space in between the asterisks for the second sub-tasks, especially at the top of the triangle. For the right-angled triangle, we only had to care about the length of the hashtag string. The first sub-task was simply implemented by reading the desired length of the triangle and making sure it was a greater value than zero. If it was, then we would go about to print the triangle recursively with a counter to ensure that the number of columns was printed properly. We also had a separate print-helper to go to the next row.

2. Console Output vs. File Output

Regarding the console output versus the file output, the console output was relatively easy to implement by using the write\1 function to properly output the hashtag rows into the console. However, the other sub-task, we instead use the write\2 function to write the parameterized lines into the parameterized file, with the file and string being generated beforehand. Several challenges arose regarding understanding how to properly use the functions, but with the help of documentation and resources, we were able to properly handle its behaviors.

3. User Input Handling

In order to properly handle inputs coming in from the console, we simply used the read\1 function to grab the value needed and to use it to properly print out the length of the right-angle triangle into the console itself. This was also done by ensuring that the input was an integer that would then be stored into its own variable. The filename was simply prompted to create the file itself, which would then be written into for the isosceles triangle.

4. Recursion and Pattern Generation

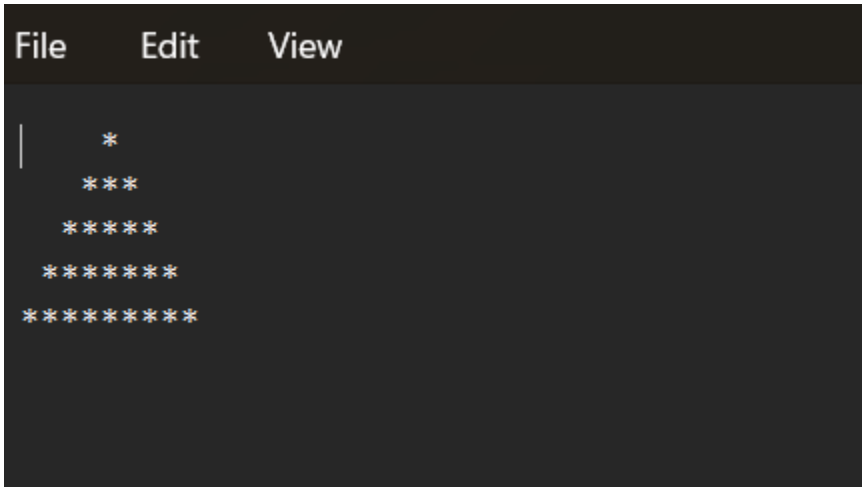
The recursive logic was simply based on the column and the length of the triangle when printed. When the height is greater or equal to the number of columns is when the row would be printed, at which, the print button would write out a specific number of hashtags before going to a newline. The second task sees us doing something similar, but instead with the appending of a line of spaces and asterisks for each newline for each line within the file.

5. Testing and Sample Outputs

```
?- right_angle_triangle_console.
Enter the height of the right-angled triangle: 5
|: .

#
##
###
####
#####
false.

?-
```



From inputting isosceles_triangle_pattern_file(5, 'triangle.txt').

Task B:

1. DCG Explanation:

From the implement DCG, we have the following types and grammars:

- **Character Type:**
  - Defines the two main types of characters; either **hero** or **enemy.**
    - Characters can belong to either category within the DCG.

```
character_type(hero) --> [hero].
character_type(enemy) --> [enemy].
```

- **Character Subtype:**
  - Within the two character types, each category has a subtype to deeper describe their primary role.
  - These subtypes are constrained under the member/2 predicate.

```
character_subtype(hero, Subtype) --> [Subtype], { member(Subtype, [wizard, mage, elf]) }.
character_subtype(enemy, Subtype) --> [Subtype], { member(Subtype, [darkwizard, demon, basilisk]) }.
```

- **Sequence:**
  - Representing an identifier for certain characters

```
sequence(Sequence) --> [Sequence], { integer(Sequence), Sequence > 0 }.
```

- **Movement Direction**
  - Defines the movement of the character, whether they move forward and backwards depending on whether they have a weapon.

```
movement_direction(hero, away, no_weapon) --> [away].
movement_direction(hero, towards, has_weapon) --> [towards].
movement_direction(enemy, towards, no_weapon) --> [towards].
```

- **Health**
  - Identifier that specifies whether they have a good or weak amount of health on varying levels.
  - Simple array of levels to show the strength of the entity.

```
health(very_weak) --> [very_weak].
health(weak) --> [weak].
health(normal) --> [normal].
health(strong) --> [strong].
health(very_strong) --> [very_strong].
```

- **Weapon**
  - Identifier showing whether the entity has a weapon or not.

```
weapon(hero, Weapon) --> [Weapon], { member(Weapon, [has_weapon, no_weapon]) }.
weapon(enemy, no_weapon) --> [no_weapon].
```

- **Movement Style**
  - Describe the entity's behaviour in movement.
    - Jumpy, smooth or stealthy.

```
movement_style(stealthy) --> [stealthy].
movement_style(smoothly) --> [smoothly].
movement_style(jerky) --> [jerky].
```

2. Testing and Results:

```
?- phrase(character_description, [enemy, demon, 7, towards, strong, no_weapon, stealthy]).
true
```

Passes, as each of the conditions match the enemy behaviour.

```
?-
|    phrase(character_description, [hero, wizard, 12, towards, normal, has_weapon, smoothly
]).
true
```

Passes, as each of the conditions match the hero's behaviour.

```
?- phrase(character_description, [hero, elf, 5, towards, very_strong, no_weapon, jerky]).
false.
```

Fails, as the hero moves towards the enemy, but has no weapon.

3. Challenges:

Similar to the idea of building a struct, and similar to the Binary Grammar we learnt in Programming Language Concepts, a DCG allows us to simulate the behavior of that structure to ensure restrictions on the character based on their character type. This structure overall allows us to create intuitive rules that abide by Prolog's logical programming concept.

Task C:

1. Explanation of Dynamic Predicates

In Prolog, dynamic predicates are used to create custom changes to the predicates behavior at runtime, which in this case, is to add, remove or modify the stored information. In our task, the dynamic predicates, book/4 and borrowed/4 are used to keep track of the books that are available and the status of each of the books to make sure we have an updated list of the books at runtime.

- book/4:
    - Single predicate represents a single book in the database, which has the following parameters:
        - Title
        - Author
        - Year
        - Genre
    - Used to maintain a list of books in a dynamic fashion.
    - Has the functions of add_book/4 and remove_book/4 to change the list of books dynamically.
        - add_book/4 adds a book to the dynamic predicate. Also checks if the book already exists.
        - remove_book/4 removes a book to the dynamic predicate.
- borrowed/4:
    - Same idea as books/4, but is used to represent the borrowing status of books.
    - Has the same parameters as book/4, but has functions that are used to check and change the list of available books.
        - is_available/4 checks the availability of a book

- borrow_book/4 marks the book as borrowed. Checks if the book is available and uses the assertz function,
- return_book/4 returns the book to unlabel it as borrowed. Uses the retract function.

2. Explanation of Each Predicate: Provide a brief explanation of how each predicate (add, remove, borrow, return, etc.) operates within the library system.

All functions have been explained above, except for the following functions:

- find_by_author/2:
  - Finds different books within the knowledge base based on a specified author. Uses the findall function with the specific author.
- find_by_genre/2:
  - Finds different books within the knowledge base based on a specific genre. Uses the findall function with the specific genre.
- find_by_year/2:
  - Finds different books within the knowledge base depending on the parameterized year. Uses the findall function with the specific year to return the title from the books dynamic predicate.
- recommend_by_genre/2:
  - Uses the find_by_genre/2 function to recommend a list of books based on the genre.
- recommend_by_author/2:
  - Uses the find_by_author/2 function to recommend a list of books based on the desired author.

3. Testing Evidence:

```
?- 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.

?-
```

Adding initial books to the library.

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'].
```