

Lab 7

CSI 3120 - Programming Language Concepts

Fall 2024

University of Ottawa

Course Coordinator: Dr. Omar Badreddin

Group 40

Anas Taimah #300228842

Omar Abdul #300228700

Yusuf Khan #300293842

Due Date: November 21th, 2024

TASK A: Generating Patterns with User Input

1. Objective and Approach

Purpose of Task A

The purpose of this task was to generate two different types of triangle patterns using Prolog: a right-angled triangle, printed to the console, and an isosceles triangle, written to a file. These tasks allowed us to explore Prolog's recursion capabilities, user input handling, and file I/O operations.

Implementation Approach

For the right-angled triangle in Question 1, we implemented a recursive predicate, `print_triangle/2`, which starts with the first row and increments until the desired height is reached. Each row is generated using a helper predicate, `print_row/2`, which recursively prints `#` symbols.

For the isosceles triangle in Question 2, we wrote a predicate, `isosceles_triangle_pattern_file/2`, that calculates the number of spaces and stars for each row and writes the pattern to a specified file. We used the predicates `write_spaces/2` and `write_stars/2` to handle formatting, and `open/3` and `close/1` to manage the file stream.

2. Console Output vs. File Output

Right-Angled Triangle

To print the right-angled triangle, we used `write/1` to display the `#` symbols row by row in the console. After prompting the user for the height with `read/1`, the `print_triangle/2` predicate recursively generated each row, adding one more `#` with each iteration. We ensured each row was properly formatted by using `nl` for newlines between rows.

Isosceles Triangle

To write the isosceles triangle to a file, we calculated the required spaces and stars for each row using the predicates `write_spaces/2` and `write_stars/2`. These were combined in the recursive predicate `write_triangle_pattern/3`, which writes each row to the file stream. We managed the file using `open/3` to create the stream and `close/1` to finalize the operation. This approach allowed us to format and save the triangle accurately.

3. User Input Handling

For both tasks, we prompted the user for input using `write/1` and `read/1`. In Question 1, the user provided the height of the triangle, which we validated to ensure it was a positive integer. Similarly, in Question 2, the user specified both the height and the output filename. We handled the input directly in the main predicates and ensured it was passed correctly to the recursive procedures.

4. Recursion and Pattern Generation

Recursive Logic

The recursive logic for both patterns follows a similar approach but differs in structure:

- For the right-angled triangle, we recursively generated rows with increasing lengths, stopping when the row count exceeded the specified height.
- For the isosceles triangle, we calculated the spaces and stars for each row dynamically. The recursive predicate continued until all rows were written to the file.

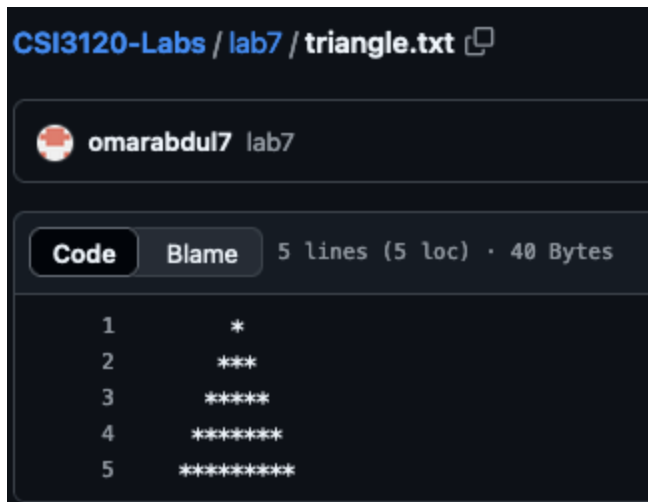
Base and Recursive Cases

The base cases for both tasks stop recursion when the current row exceeds the specified height. The recursive cases handle the construction of each row, incrementing the row count and re-evaluating the necessary parameters.

5. Testing and Sample Outputs

```
?- right_angle_triangle_console.  
Enter the height of the right-angled triangle: -4.  
Error: Height must be a positive integer.  
false.  
  
?- right_angle_triangle_console.  
Enter the height of the right-angled triangle: 4.  
  
#  
##  
###  
####  
true.  
  
?- isosceles_triangle_pattern_file(-5, 'triangle.txt').  
Error: Height -5 must be a positive integer.  
false.  
  
?- isosceles_triangle_pattern_file(5, 'triangle.txt').  
Isosceles triangle pattern written to file: triangle.txt  
true
```

Here, we can see the console output for some sample input. Our code covers error handling for when a user inputs a negative integer.



The screenshot shows a code editor interface with a dark theme. At the top, the file path is 'CSI3120-Labs / lab7 / triangle.txt'. Below the path, the username 'omarabdul7' and the file name 'lab7' are displayed. The editor shows a code block with 5 lines of code, which is 5 lines (5 loc) and 40 Bytes. The code block is titled 'Code' and 'Blame'. The code itself is as follows:

```
1      *
2     ***
3    *****
4   *
4   *****
5  *
5  *****
```

This is the file output for the isosceles triangle pattern question for a given user input 5, and the file name *triangle.txt*.

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

1. DCG Explanations

The main rule `character_description` combines different parts of a character, like `character_type`, `character_subtype`, `sequence`, `movement_direction`, `health_level`, `weapon`, `valid_movement`, and `movement_style`. Characters also can have two different roles, either enemy or hero. The enemy role has different types of sub roles such as dark wizard, demon, and basilisk, while the hero role has sub roles such as wizard, mage, and elf. Each character also has a sequence number which has constraints which force the number to be positive. The different movement directions are defined as “towards” and “away”, as well as different health levels and weapon possession rules are defined. The character's movement direction depends on the character's role and their weapon possession. Enemies will move “towards” no matter what, on the other side, heroes with weapons will always move “towards”, and heroes without weapons will always move “away”.

2. Testing and Results

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

true.

This test passed in our code as all the rules which concerned the enemy role were followed.

- Sub role demon is part of enemy role
- 7 is positive
- Moves towards

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

true.

This test passed in our code as all the rules which concerned the hero role were followed.

- Sub role wizard is part of hero role
- 12 is positive
- Moves towards since it has a weapon

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

false.

This test failed in our code as all the rules which concerned the hero role were followed.

- Sub role elf is part of hero role
- 5 is positive
- (fails) Moves towards without a weapon

3. Challenges

One of the main challenges we faced was making sure that movement direction depended on both the character role (hero or villain) and whether they had a weapon. We created a separate rule called `valid_movement` which checks these conditions to see if it's a valid movement. The other challenge was making sure that sequence number was a positive integer, which needed extra code to properly function.

Task C: Library Management System with Prolog

1. Explanation of Dynamic Predicates

Dynamic predicates allow the programs to make modifications during runtime. The code declares the predicates `book/4` and `borrowed/4` as dynamic, so they can be updated during the programs runtime. The `book/4` predicate represents books in the library, keeping its information such as its title, author, publication year, and genre. The `borrowed/4` predicate tracks borrowed books with the same information. Dynamic predicates help the program maintain the library state easier than if it was static.

2. Explanation of Each Predicate

`initialize_library/0`: Clears all books and borrowed books from the library in order to reset the system

`add_book/4`: Adds a new book to the library with the given information. It filters out duplicate books to keep the library unique

`remove_book/4`: Removes the requested book from the library even if it is being borrowed

`is_available/4`: Checks if a book is available for borrowing if its not currently being borrowed

`borrow_book/4`: Adds an entry to the `borrowed/4` predicate with the book info

`return_book/4`: Removes a book entry from the `borrowed/4` predicate

`find_by_author/2`: Finds all books by a specific author

`find_by_genre/2`: Finds all books in a specific genre

`find_by_year/2`: Finds all books published in a specific year

`recommend_by_genre/2`: Gives a list of book recommendations based on a genre.

`recommend_by_author/2`: Gives a list of book recommendations based on an author.

`list_books/0`: Lists all the books in the library

`print_books/1`: Helper predicate to format the print statements for books.

3. Testing Evidence

Example:

```
add_book('Harry Potter', 'J.K Rowling', '1997', 'Fiction')
true.
```

```
list_books
```

```
Title: Harry Potter, Author: J.K Rowling, Year: 1997, Genre: Fiction
```

```
is_available('Harry Potter', 'J.K Rowling', '1997', 'Fiction')  
true.
```

```
borrow_book('Harry Potter', 'J.K Rowling', '1997', 'Fiction')  
true.
```

```
is_available('Harry Potter', 'J.K Rowling', '1997', 'Fiction')  
false.
```

```
return_book('Harry Potter', 'J.K Rowling', '1997', 'Fiction')  
true.
```

```
is_available('Harry Potter', 'J.K Rowling', '1997', 'Fiction')  
true.
```

```
find_by_author('J.K Rowling', Books)  
Books = ['Harry Potter'].
```

```
find_by_genre('Fiction', Books)  
Books = ['Harry Potter'].
```

```
recommend_by_genre('Fiction', Recommendations)  
Recommendations = ['Harry Potter'].
```

ChatGPT declaration:

Our group used ChatGPT throughout the lab to better understand the program application so that we are able to develop our answers. ChatGPT was used for assistance to learn and understand how it worked and not to generate any answers or results. All solutions were implemented in our own words and approach.