CSI 3120

Lab 6 Report

Group #6

Carl Li 300235679
Ruoyu Liu 300176134
Roy Rui 300176548

Task1

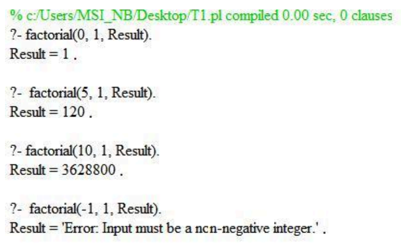
Question 1: Write a tail-recursive Prolog predicate to compute the factorial of a non-negative integer using an accumulator. The predicate should validate the input to ensure it is a non negative integer and return an error message for invalid inputs (e.g., negative numbers or non integer values). Handle invalid input: Ensure that the input is a valid non-negative integer, returning an appropriate error message for invalid inputs (e.g., negative numbers or non-integer values). Add multiple base cases: Use a base case for `n = 0` and additional validation for `n < 0` (invalid case).

Explanation:

1. Factorial with Tail Recursion and Accumulators:
The accumulator is used to store the intermediate results during the recursive process.
The predicate factorial_helper/3 is used to calculate the factorial. The accumulator Acc keeps track of the product calculated so far. Initially, Acc is set to 1, and the factorial computation proceeds by multiplying Acc with the current value of N in each recursive call.
The base case is when N reaches 0, at which point the result is Acc, which holds the final value of the factorial.
This use of an accumulator allows the recursion to be **tail-recursive.**

Input Validation:
Number(N), integer(N), N>=0, if the predication doesn't meet, return Result = 'Error: Input must be a non-negative integer.'

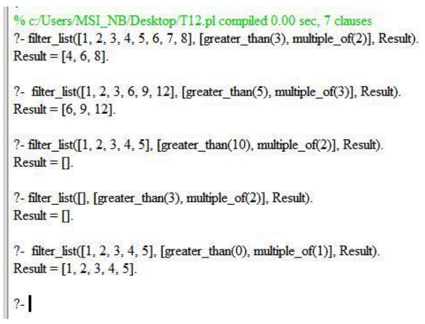
Output:



```
% c:/Users/MSI_NB/Desktop/T1.pl compiled 0.00 sec, 0 clauses
?- factorial(0, 1, Result).
Result = 1 .

?-  factorial(5, 1, Result).
Result = 120 .

?- factorial(10, 1, Result).
Result = 3628800 .

?-  factorial(-1, 1, Result).
Result = 'Error: Input must be a ncn-negative integer.' .
```

Question 2: Write a Prolog predicate filter_list/3 that filters elements from a list based on dynamically passed conditions. The conditions will be passed as a list of predicates, and the result should include only the elements that satisfy all the conditions. Define a Predicate filter_list/3: Filters a list based on the two conditions

Output:



```
% c:/Users/MSI_NB/Desktop/T12.pl compiled 0.00 sec, 7 clauses
?- filter_list([1, 2, 3, 4, 5, 6, 7, 8], [greater_than(3), multiple_of(2)], Result).
Result = [4, 6, 8].

?-  filter_list([1, 2, 3, 6, 9, 12], [greater_than(5), multiple_of(3)], Result).
Result = [6, 9, 12].

?- filter_list([1, 2, 3, 4, 5], [greater_than(10), multiple_of(2)], Result).
Result = [].

?- filter_list([], [greater_than(3), multiple_of(2)], Result).
Result = [].

?-  filter_list([1, 2, 3, 4, 5], [greater_than(0), multiple_of(1)], Result).
Result = [1, 2, 3, 4, 5].

?-
```

Explanation :

2. Filtering Elements Based on Multiple Conditions:

```
greater_than(Range, Element) :-
```

Checks if the Element is a number and is greater than a given Range.

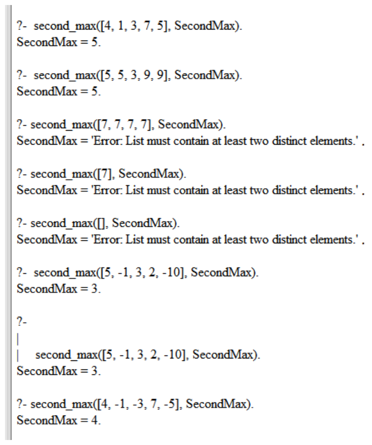
```
multiple_of(Divisor, Element) :-
    number(Element),
    Divisor > 0,
    0 is Element mod Divisor.
```

Used to check if the Element is a number and is a multiple of the given Divisor. The dynamic part here is that the divisor can be any value greater than 0.

Recursive Logic:
filter_list(List, Conditions, Result) is the main predicate that calls the auxiliary predicate filter_helper/4 to do the actual filtering, using the accumulator (Acc).
filter_helper/4 is a recursive predicate that processes each element in the list. It takes the head of the list (H), the list of conditions, and the accumulator (Acc), and checks if the head satisfies all the conditions.
The predicate check_all_conditions(H, Conditions) is called to verify if the element (H) satisfies all the conditions. If all conditions are satisfied, the element (H) is added to the accumulator ([H|Acc]), and processing continues recursively with the tail (T). If the conditions are not satisfied, the element is skipped and processing continues recursively with the tail (T).

Question 3: Write a Prolog predicate `second_max/2` that finds the second maximum element in a list. The predicate should handle the following cases: Lists with duplicate values (e.g., if the maximum value appears multiple times, it should still find the second distinct maximum). Lists with fewer than two distinct elements should return an appropriate error message.

Output:



```
?-  second_max([4, 1, 3, 7, 5], SecondMax).
SecondMax = 5.

?-  second_max([5, 5, 3, 9, 9], SecondMax).
SecondMax = 5.

?- second_max([7, 7, 7, 7], SecondMax).
SecondMax = 'Error: List must contain at least two distinct elements.' .

?- second_max([7], SecondMax).
SecondMax = 'Error: List must contain at least two distinct elements.' .

?- second_max([], SecondMax).
SecondMax = 'Error: List must contain at least two distinct elements.' .

?-  second_max([5, -1, 3, 2, -10], SecondMax).
SecondMax = 3.

?-
|
|    second_max([5, -1, 3, 2, -10], SecondMax).
SecondMax = 3.

?- second_max([4, -1, -3, 7, -5], SecondMax).
SecondMax = 4.
```

Explanation:

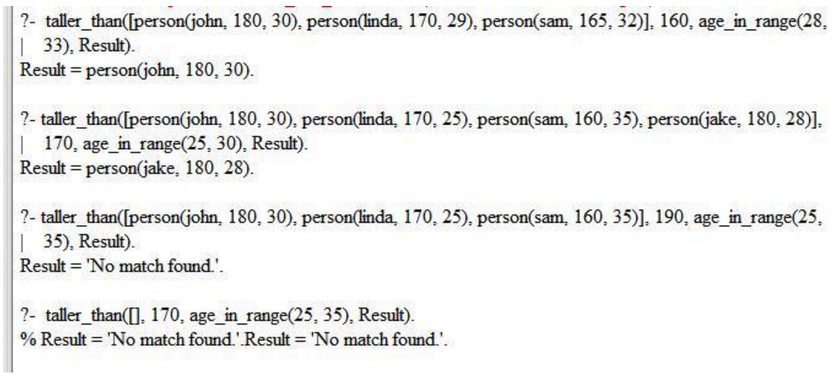
3. Finding the Second Maximum Element:
Handling Duplicates: To ensure that duplicate values do not interfere with finding the second-largest element, the input list is converted to a set using the list_to_set/2 predicate. This removes duplicate values and makes each element unique in the set.
In second_max/3, the input list is first converted to a set using list_to_set/2, and then a length check is performed to ensure that there are at least two unique elements. This ensures that the second-largest element is found correctly in the end, even if there are duplicate values in the list.

Error Handling for Insufficient Elements: The predicate second_max(List, 'Error: List must contain at least two distinct elements.') It is used to handle the case where there are not enough elements in the list.
First, the list length is checked to see if it is less than 2 using length (List, Len). If so, an error message is returned.
Next, the list is converted to a set (using list_to_set/2) and the length is checked again to ensure that there are at least two distinct elements.

Task 2

You are given a list of people, each represented by their name, height, and age (e.g., person(Name, Height, Age)). Your task is to write a Prolog predicate `taller_than/4` that finds the first person in the list who is taller than a specified height and age between a minimum (Inclusive) and maximum (Exclusive) range. Once a match is found, the search should stop, and no further solutions should be explored. Use the cut operator (`!`) to ensure that backtracking halts after finding the first person who satisfies both conditions. Ensure that your solution can handle edge cases, such as an empty list or when no one meets both conditions.

Output:



```
?-  taller_than([person(john, 180, 30), person(linda, 170, 29), person(sam, 165, 32)], 160, age_in_range(28,
|   33), Result).
Result = person(john, 180, 30).

?- taller_than([person(john, 180, 30), person(linda, 170, 25), person(sam, 160, 35), person(jake, 180, 28)],
|   170, age_in_range(25, 30), Result).
Result = person(jake, 180, 28).

?- taller_than([person(john, 180, 30), person(linda, 170, 25), person(sam, 160, 35)], 190, age_in_range(25,
|   35), Result).
Result = 'No match found.'.

?-  taller_than([], 170, age_in_range(25, 35), Result).
% Result = 'No match found.'.Result = 'No match found.'.
```

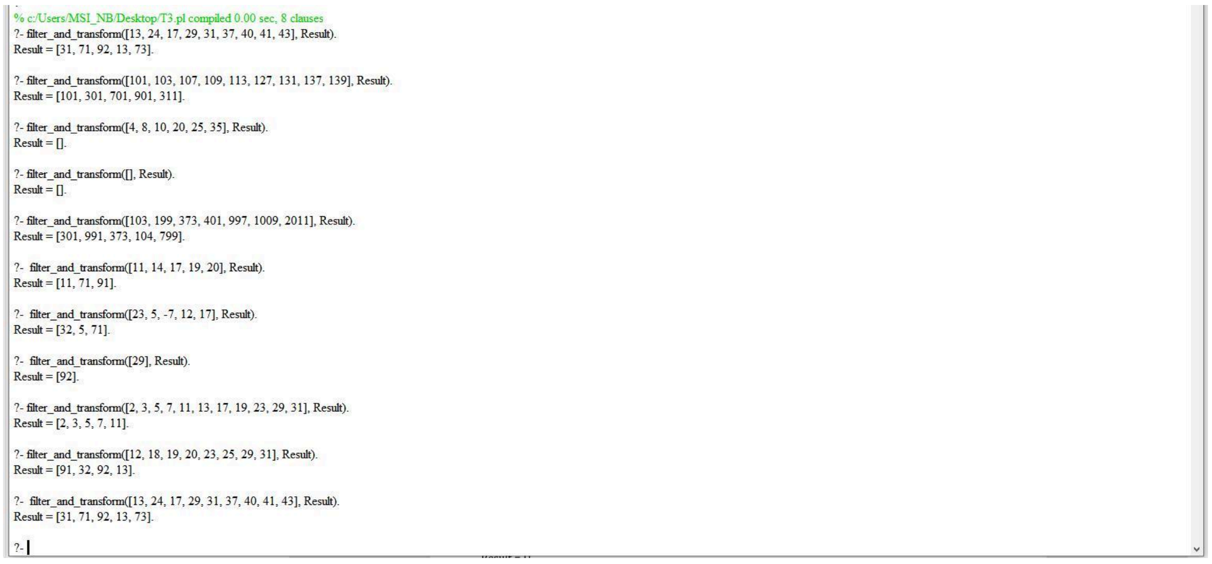
Explanation:

1. Explain how you implemented backtracking control using the cut (`!`) operator to stop the search after finding the first valid match.
The cut operator (!) is used in the taller_than/4 predicate to stop backtracking. Once the first matching condition is found, ! is called, preventing backtracking to find other possible solutions.
2. Describe how your code processes the input list of `person(Name, Height, Age)` to apply both the height and fixed age conditions.
The code processes the input list containing person(Name, Height, Age) through the find_person/5 predicate.
For each element, it checks whether the height exceeds the minimum height using Height > MinHeight, and whether the age is within the given range using Age >= MinAge and Age =< MaxAge.
If an element satisfies all the conditions, it is returned as a Result.

3. Explain how you handled errors and edge cases, such as empty lists or cases where no valid match is found.
If the input list is empty ([]), then taller_than([], _, _, 'No match found.') is called:- !. This returns 'No match found.', and backtracking is stopped using the cut operator.

Task 3

Write a Prolog predicate filter_and_transform/3 that: 1. Filters elements from a list by checking if each number is a prime number. 2. Transforms the filtered prime numbers by reversing the digits of each number. 3. Ensures tail-recursive optimization using an accumulator. 4. Uses the cut operator (!) to stop the search after exactly 5 matching prime numbers have been found and transformed.



```
% c:/Users/MSI_NB/Desktop/T3.pl compiled 0.00 sec, 8 clauses
?- filter_and_transform([13, 24, 17, 29, 31, 37, 40, 41, 43], Result).
Result = [31, 71, 92, 13, 73].

?- filter_and_transform([101, 103, 107, 109, 113, 127, 131, 137, 139], Result).
Result = [101, 301, 701, 901, 311].

?- filter_and_transform([4, 8, 10, 20, 25, 35], Result).
Result = [].

?- filter_and_transform([], Result).
Result = [].

?- filter_and_transform([103, 199, 373, 401, 997, 1009, 2011], Result).
Result = [301, 991, 373, 104, 799].

?-  filter_and_transform([11, 14, 17, 19, 20], Result).
Result = [11, 71, 91].

?-  filter_and_transform([23, 5, -7, 12, 17], Result).
Result = [32, 5, 71].

?-  filter_and_transform([29], Result).
Result = [92].

?- filter_and_transform([2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31], Result).
Result = [2, 3, 5, 7, 11].

?- filter_and_transform([12, 18, 19, 20, 23, 25, 29, 31], Result).
Result = [91, 32, 92, 13].

?-  filter_and_transform([13, 24, 17, 29, 31, 37, 40, 41, 43], Result).
Result = [31, 71, 92, 13, 73].

?- 
```

Explanation :

1. Describe the logic behind the prime-checking predicate and how it ensures that only prime numbers are selected.
is_prime(N) first checks whether N is greater than 2, and then uses \+ has_factor(N, 2) to ensure that no factors divisible by N are found in the iterations starting from 2.

2. Discuss the use of any helper predicates that handle the digit-reversing process.
reverse_digits/2 is an auxiliary predicate that reverses the number of digits in a number. First, the number N is converted to a list of characters using number_chars(N, Chars) and then the character list is reversed using reverse/2. Finally, the reversed character list is converted back to a number.

3. Explain how your code uses Prolog's backtracking mechanism to stop after exactly 5 prime numbers are found and transformed.

In the filter_and_transform/4 predicate, an accumulator and tail recursion are used to process the input list.
When 5 prime numbers are found or the input list is exhausted, filter_and_transform(_, Acc, 5, Result) is called:- !, reverse(Acc, Result), and cut (!) is used to stop backtracking.

Reference:

1. At first, we used the traditional method to determine the correctness of the input value and returned an Error if the input value was incorrect.
factorial(N, Result) :- (\+ number(N) ; \+ integer(N) ; N < 0), !, write('Error: Input must be a non-negative integer.'), nl, fail.
We asked GPT if there was a simpler way to determine the error, and learned that we could directly add the error information to the result of the method to combine the check.
factorial(N, _, 'Error: Input must be a non-negative integer.') :-

2. We ask ChatGPT for the dynamical condition In Prolog,
GPT transcription：you can dynamically pass conditions to perform different calculations or logical judgments. This feature is very suitable for scenarios where you need to filter lists by conditions or process them according to multiple different rules provided by the user. call/2 is a powerful built-in predicate that can be used to call dynamic conditions passed in by other predicates. You can pass a predicate as an argument and call it when needed.

3. We asked for the logic of multiple_of/2 function：The condition is true if Element is a number, Divisor is greater than 0, and Element is divisible by Divisor.

4. In task 2, the logic of recursive part is assistant by ChatGPT to better combing structure. age_in_range(MinAge, MaxAge) nicely expresses the specified height and age between a minimum (Inclusive) and maximum (Exclusive) range

5. In task 3, We asked how to check prime numbers. GPT gave two ways. One is to create a function has_factor(N, F) to find out whether N has a divisible factor, so as to determine whether N is a prime number.
The logic of is_prime/1 is:
If N is greater than 2 and there is no factor from 2 to sqrt(N), N is considered prime.
has_factor/2 checks whether N is divisible by F by incrementing from F=2.
If a factor is found during the iteration, it proves that N is not a prime number. Another way is to create a has_divisor/2 whose logic is to check whether a given number N has a factor Divisor. If there is a Divisor that can divide N, it means that N is not a prime number. We choose the former method.

6. Used GPT to help correct the logic of the recursive part. If H is prime:
Use reverse_digits(H, Reversed) to reverse the digits.