

LAB7-VARIANT2 GROUP 12 REPORT SHEET #Ayşe Nur Gümüş- Kaan Baturalp Coşdan

Project Scope: This project is a Prolog program that calculates the number of days between two dates. The program represents dates in the format **date(Year, Month, Day)** and assumes the year to be 2023. The project has the capability to calculate the correct number of days between dates with varying month and day values.

Logic Flow and Algorithm: The core algorithm of the project utilizes the predicate `calculate_days/5`. This predicate works in conjunction with other helper predicates to calculate the number of days between two dates. Here is the logic of the algorithm:

- 1-) Two dates, `Date1` and `Date2`, are taken as input.
- 2-) Extract the year, month, and day information from `Date1` and `Date2`.
- 3-) The `calculate_days/5` predicate operates based on different scenarios:
 - a. If the two dates are the same, return 0 as the number of days.
 - b. If `Date1` comes before `Date2`, calculate the number of days.
 - c. If `Date1` and `Date2` represent the same month, calculate the number of days directly. Call the predicate `calculate_days/5`, passing `Date1`, `Date2`, and an initial day count of 0.
- 4-) If `Date1` and `Date2` represent the same month, calculate the number of days directly as `Date2 - Date1`.
- 5-) The result is stored in the `Days` variable and returned.

Main Components of the Code:

The main components of the project's code include:

`days_in_month/2`: A rule mapping month numbers to the corresponding number of days in that month.

`calculate_days/5`: The main rule that calculates the number of days between two dates. It works in conjunction with helper rules.

Helper rules:

If the two dates are the same, return 0 as the number of days.

If the first date comes before the second date, calculate the number of days.


If the two dates represent the same month, calculate the number of days directly.

These components come together in the code to perform date calculations and produce accurate results.

OUTPUTS

We had observed some significant days ;


Month change (January 31st to February 1st):

 `days_between_dates(date(2023, 1, 31), date(2023, 2, 1), Days).`

Days = 1

?- days_between_dates(date(2023, 1, 31), date(2023, 2, 1), **Days**).


Within the same month (March 15th to March 20th):

 `days_between_dates(date(2023, 3, 15), date(2023, 3, 20), Days).`

Days = 5

?- days_between_dates(date(2023, 3, 15), date(2023, 3, 20), **Days**).


Across different months (April 10th to May 15th):

 `days_between_dates(date(2023, 4, 10), date(2023, 5, 15), Days).`

Days = 35

?- days_between_dates(date(2023, 4, 10), date(2023, 5, 15), **Days**).


Boundaries (January 1st to December 31st):

 `days_between_dates(date(2023, 1, 1), date(2023, 12, 31), Days).`


Days = 364

?- days_between_dates(date(2023, 1, 1), date(2023, 12, 31), **Days**).

Overview

 `days_between_dates(date(2023, 3, 15), date(2023, 3, 20), Days).`

Days = 5

 `days_between_dates(date(2023, 1, 31), date(2023, 2, 1), Days).`

Days = 1


Next

10

100

1,000

Stop

 `days_between_dates(date(2023, 4, 10), date(2023, 5, 15), Days).`

Days = 35


Next

10

100

1,000

Stop

 `days_between_dates(date(2023, 1, 1), date(2023, 12, 31), Days).`

Days = 364

Next

10

100

1,000

Stop

Comments

If the algorithm had to consider leap years (e.g., 2020, 2024, 2028, etc.), some modifications would be required. However, since you are working only with the year 2023, focusing on that specific year made the project more straightforward. By testing the critical dates, you were able to obtain accurate results. Working with Prolog interface provided a great convenience for observing the outcomes.