Lab 7. EARIN

Variant 1

Oleg Kim

Seniv Volodymyr

02.06.2023

Task description:

Perform simple addition and subtraction of dates by a number of days.

Assumptions:

- The year is 2023.
- N <= 365.

Theoretical background:

Prolog, short for "Programming in Logic," is a logic programming language widely used in artificial intelligence and computational linguistics. It was developed in the 1970s by Alain Colmerauer and his colleagues as a tool for natural language processing. Prolog is based on formal logic and provides a declarative approach to problem-solving.

Theoretical Background:

1. Logic Programming: Prolog is rooted in logic programming, which is based on the idea of representing knowledge and solving problems using formal logic. It is founded on the logical concept of Horn clauses, which are logical implications in the form of "if-then" statements.
2. Predicate Logic: Prolog is based on predicate logic, which is an extension of propositional logic. It allows for the representation of complex statements using predicates, which are relationships between objects or entities. In Prolog, predicates are defined as logical rules and facts.
3. Facts and Rules: Prolog programs consist of a collection of facts and rules. Facts represent simple statements or assertions about the relationships between objects. Rules define logical implications and provide the means for logical inference. Prolog uses these facts and rules to derive new information or answer queries.
4. Unification: Prolog employs a process called unification, which is fundamental to its operation. Unification is the process of matching and binding variables to values, enabling the evaluation of logical statements. It allows Prolog to search for solutions by finding variable assignments that satisfy the given logic.

Launch instruction:

If you want to start the code, you must use Prolog and its online environment “swi-prolog” for quick experiments with no installation.

https://swish.swi-prolog.org/

Solution:

Our application just adds and subtracts provided number of days, from the date the user provides, and returns the result date.

Challenging parts:

We had some problem with handling the leap-year and the cases where we are adding or subtracting the number of days which lead to increase or decrease the year, but we released it successfully.

The basic logic is as follows:

The overall logic flow of the solution involves converting between dates and timestamps, performing arithmetic operations on dates using timestamps, validating dates, and utilizing the facts for the number of days in each month.

Code overview:

date_to_timestamp(Date, Timestamp): This predicate converts a given Date to a Timestamp using the date_time_stamp/2 built-in predicate.

timestamp_to_date(Timestamp, Date): This predicate converts a given Timestamp to a Date using the stamp_date_time/3 built-in predicate. The Timestamp is first converted to a DateTime in UTC, and then the date component is extracted.

add_date(Date, DaysToAdd, NewDate): This predicate adds DaysToAdd number of days to a given Date to calculate the NewDate. It converts the Date to a Timestamp, adds the appropriate number of seconds (converted from days), and then converts the resulting NewTimestamp back to a NewDate using timestamp_to_date/2. It also ensures that the year of the NewDate is either 2023 or 2024.

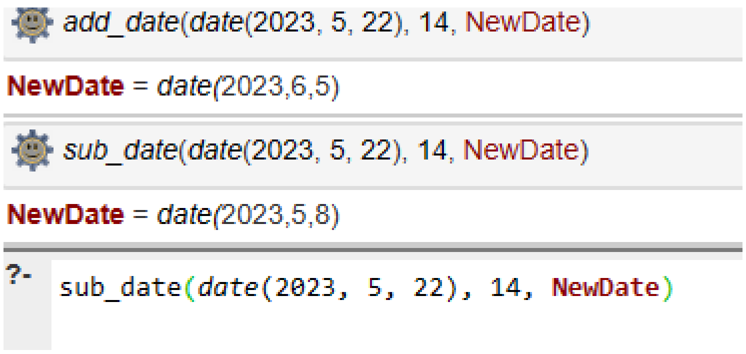
sub_date(Date, DaysToSubtract, NewDate): This predicate subtracts DaysToSubtract number of days from a given Date to calculate the NewDate. Similar to add_date/3, it converts the Date to a Timestamp, subtracts the appropriate number of seconds (converted from days), and then converts the resulting NewTimestamp back to a NewDate using timestamp_to_date/2. It also ensures that the year of the NewDate is either 2022 or 2023.

validate_date(date(Year, Month, Day)): This predicate validates a given date/3 term. It checks that the Year is 2023 and the Month is between 1 and 12. It also uses the days_in_month/2 facts to verify that the Day is within the valid range for the given Month.

days_in_month(Month, DaysInMonth): This is a set of facts providing the number of days in each month. It associates a given Month with the corresponding DaysInMonth.

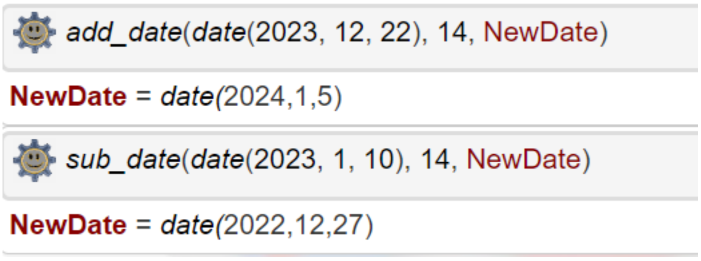
Result:

Simple subtraction and addition:



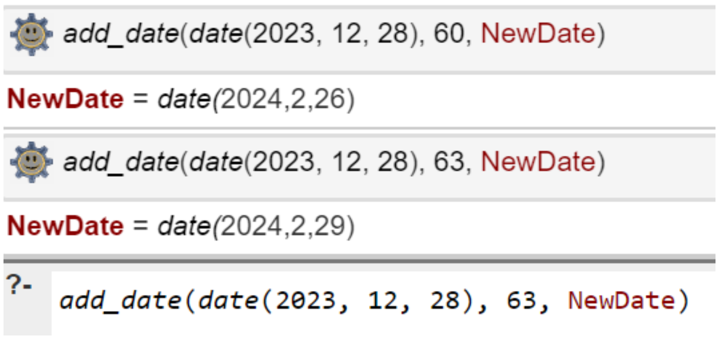
```
add_date(date(2023, 5, 22), 14, NewDate)
NewDate = date(2023,6,5)
sub_date(date(2023, 5, 22), 14, NewDate)
NewDate = date(2023,5,8)
?- sub_date(date(2023, 5, 22), 14, NewDate)
```

Subtraction and addition with increasing and decreasing the year:



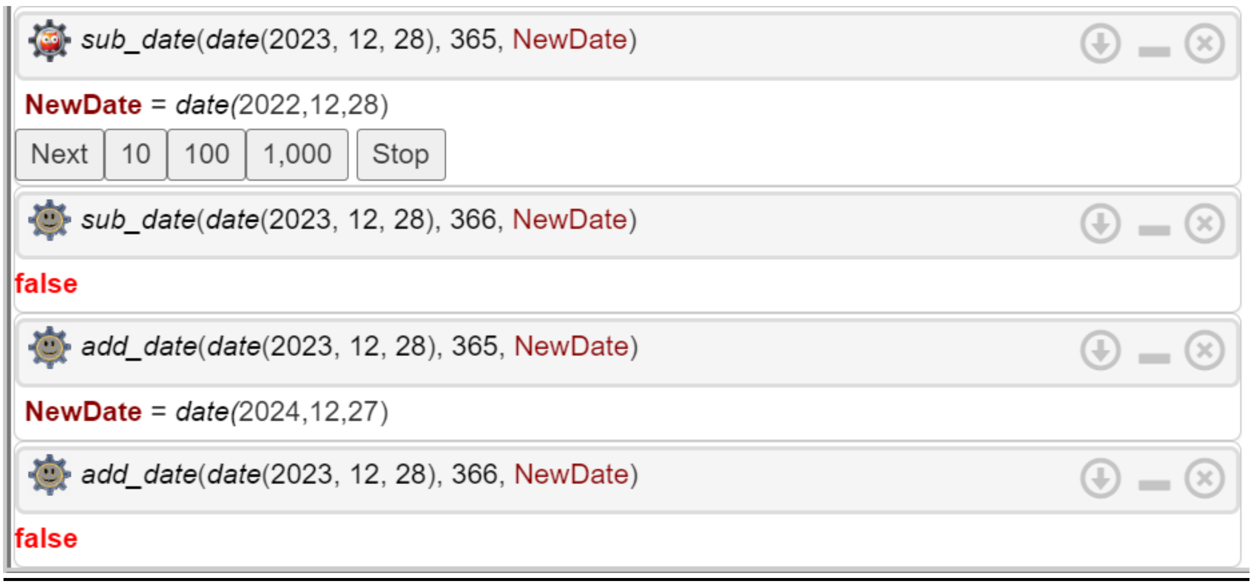
```
add_date(date(2023, 12, 22), 14, NewDate)
NewDate = date(2024,1,5)
sub_date(date(2023, 1, 10), 14, NewDate)
NewDate = date(2022,12,27)
```

Working with leap-year:



```
add_date(date(2023, 12, 28), 60, NewDate)
NewDate = date(2024,2,26)
add_date(date(2023, 12, 28), 63, NewDate)
NewDate = date(2024,2,29)
?- add_date(date(2023, 12, 28), 63, NewDate)
```

Adding and subtracting more that 365:


```
sub_date(date(2023, 12, 28), 365, NewDate)
NewDate = date(2022,12,28)
```

```
sub_date(date(2023, 12, 28), 366, NewDate)
false
add_date(date(2023, 12, 28), 365, NewDate)
NewDate = date(2024,12,27)
add_date(date(2023, 12, 28), 366, NewDate)
false
```

Conclusion:

In conclusion, the provided code presents a set of predicates for working with dates and timestamps in Prolog. It offers functionality for converting dates to timestamps and vice versa, adding or subtracting days from a given date, and validating dates based on certain criteria.