Laboratory 7

Variant 5

Group 5

By Jan Szachno and Aleksandra Głogowska

---

1. Introduction

Given a day and a date, the task is to calculate the day and provide a week date after N days of working day. The assumptions are that there are only 5 working days in a week, that the year we are operating in is 2024 and that the maximal N number of days that we can add to the starting date is 366.

The program is written in Prolog, the input query should include the main adding function with two parameters, the first one being the string with the starting date in a DDMM format, the second one being the N number of working days. The environment used for the implementation is called Swish (https://swish.swi-prolog.org/).

2. Implementation Description

Logic flow of the solution

The program takes the starting date and the N number of days as input. The algorithm starts by converting the starting date typed in a string to the prolog date format, assuming that the year is always 2024. Next, the code goes to a function (resulting_date) that will be responsible for finding the date (a working day) after N days. It recursively calls itself, decrementing the count N and finding the next working day each time. It checks if the day is a weekend day, and if so, it does not count it in and goes to the next one. The loop stops if the counted days are equal to 0. In the loop, checking the next working day is done by going to another function (next_working_day), which has the weekend day checking function (is_weekend) and the one calculating the next day (next_day) included. The latter is the main function that decides to which date the program should switch. It is used to calculate the next calendar day, taking into account the number of days in each month, including the transition between months and years. Next, it converts the result date back to the DDMM format (convert_date_to_ddmm) and then checks which weekday it is (day_of_the_week and day). At the end, the program formats the results and outputs them in the console.

Main components of the code

- Converting the user input into Date data type in Prolog:

```
% Convert DDMM to day, month, and year
convert_ddmm_to_date(DDMM, date(Y, M, D)) :-
    atom_chars(DDMM, [D1, D2, M1, M2]),
    atom_concat(D1, D2, DayAtom),
    atom_concat(M1, M2, MonthAtom),
    atom_number(DayAtom, D),
    atom_number(MonthAtom, M),
    Y = 2024.
```

It was required for us to be able to perform the calculations.

- Recursive procedure to calculate the date after n working days:

```
% Calculate the resulting date after adding working days
resulting_date(StartDate, 0, StartDate).
resulting_date(StartDate, N, ResultDate) :-
    N > 0,
    next_working_day(StartDate, NextDate),
    N1 is N - 1,
    resulting_date(NextDate, N1, ResultDate).
```

This is the main part of the program. The base case for the procedure is when we want to move by 0 days. In such a case we know that the day is the same and can return it. In other cases we want to advance by one day and check if we reached the base case again.

- Calculation of next working day:

```
% Calculate the next working day
next_working_day(Date, NextDate) :-
    next_day(Date, NextDate1),
    (is_weekend(NextDate1) -> next_working_day(NextDate1, NextDate); NextDate = NextDate1).

% Check if a date is a weekend
is_weekend(date(Y, M, D)) :-
    day_of_the_week(date(Y, M, D), W),
    (W = 6; W = 7).
```

This is another very important part of the program. In this part we calculate what is the next working day from the current date. To calculate it we advance the date by a single day. If it turns out that we landed on the weekend, we try again until we reach the end of the weekend, and thus the next working day.

- Calculation of next day:

```
% Calculate the next day
next_day(date(Y, M, D), date(Y, M, NextD)) :-
    D < 28, NextD is D + 1.
next_day(date(Y, M, 28), date(Y, M, 29)) :- member(M, [1, 3, 5, 7, 8, 10, 12]).
next_day(date(Y, M, 28), date(Y, M, 29)) :- M \= 2.
next_day(date(Y, M, 29), date(Y, M, 30)) :- member(M, [1, 3, 5, 7, 8, 10, 12]).
next_day(date(Y, M, 29), date(Y, M, 30)) :- M \= 2.
next_day(date(Y, 2, 29), date(Y, 3, 1)).
next_day(date(Y, 2, 29), date(Y, 3, 1)).
next_day(date(Y, M, 30), date(Y, M, 31)) :- member(M, [1, 3, 5, 7, 8, 10, 12]).
next_day(date(Y, M, 30), date(Y, M, 31)) :- M \= 4, M \= 6, M \= 9, M \= 11.
next_day(date(Y, M, 31), date(Y, NextM, 1)) :- NextM is M + 1.
next_day(date(Y, 12, 31), date(NextY, 1, 1)) :- NextY is Y + 1.
```

To know what is the next working day, we need to know what is the next normal day.

This part of code achieves that by defining rules of transitions between dates. For example we define that if the day of the month is smaller than 28, then we may safely advance to the next day of the month since every month has at least 28 days. Later we introduce more complicated rules to deal with the fact that different months have different number of days.

- The “main” procedure:

```
% Main predicate to calculate the day and date after adding working days
n_work_days(DDMM, N) :-
    convert_ddmm_to_date(DDMM, StartDate),
    resulting_date(StartDate, N, ResultDate),
    convert_date_to_ddmm(ResultDate, ResultDDMM),
    day_of_the_week(ResultDate, W),
    day(W, DayName),
    format(atom(Result), '~w, ~w', [DayName, ResultDDMM]),
    write(Result).
```

This is the procedure that we use to run the program. It calculates the next date after n working days using the functions described earlier in the report.

3. Challenges

This is the first time, we were using the Prolog language. Its syntax and logical programming paradigm are different from imperative programming languages. Learning this and paradigm shift into the new language was difficult.

- The language has a different approach than other languages that we were studying before. Our program in Prolog relies on recursion. Getting the base and recursive cases correct was crucial to avoid infinite loops or incorrect results.

- Primarily, incorrectly passed and returned data types (through functions) were causing a lot of errors, which needed debugging. Converting between different types (atoms, strings, numbers) was confusing.

- The function that provided calculating the next day while considering months' varying lengths and leap years also turned out to be complex.

- Handling incorrect data turned out to be difficult for us. Due to lack of proficiency in the language, we found it hard to deal with dates such as 31st of February.