Laboratory 7

Variant 4 – Convert English words of a written number (up to a thousand) into numerical digits N, where N<=1000.

Lab group 105

Group 9 – Aditya Kandula, Martyna Wielgopolan

27.04.2025

I. Introduction

In this laboratory exercise, the objective was to create a Prolog program capable of converting a number written in English words into its numeric form, restricted to numbers up to 1000.

To implement and test the solution, the SWISH online Prolog environment (https://swish.swi-prolog.org/) was used, as recommended in the lab instructions.

During this lab, we practiced the basic concepts of Prolog, such as:

- Fact and rule definition,
- Recursion for list processing,
- Pattern matching and unification,
- Working without traditional loops or mutable variables.

Compared to traditional programming languages like Python or Java, Prolog emphasizes declarative logic rather than sequential instructions. Instead of telling the program how to do something step-by-step, we describe what is true, and let Prolog infer the solution through logical inference and recursion.

This required a shift in thinking — especially when designing recursive parsing mechanisms instead of using for-loops or state variables.

II. Implementation

1. Input Normalization

- The input phrase (e.g., "Three hundred and twelve") is split into words.
- All words are converted to lowercase and converted to atoms for easy matching.

2. Word Mapping

- Words are organized into separate **predicate groups**:
    - unit/2 (for numbers from 1–9),
    - special/2 (for numbers from 10–19),
    - tens/2 (for 20, 30, ..., 90),
    - multiplier/2 (for hundred and thousand).

This modular design keeps the logic clean and extensible.

3. Recursive Parsing and Accumulation

- The program recursively processes the list of words:
    - If "hundred" or "thousand" is encountered, it multiplies the preceding number.
    - If "and" appears, it is ignored.
    - Other number words are simply added to the running total.

The use of an accumulator in recursion allows building the final numeric value step-by-step.

4. Handling English Grammar

- Optional connecting words like "and" (e.g., "one hundred and five") are skipped.
- Multiplication happens immediately after encountering a multiplier, correctly reflecting English grammar rules.

5. Example Execution

Sample queries and expected results:

?- words_to_number("forty five", X).

X = 45.

?- words_to_number("three hundred and sixty seven", X).

X = 367.

?- words_to_number("one thousand", X).

X = 1000.

?- words_to_number("six hundred and ninety nine", X).

X = 699.

III. Challenges and difficulties

While implementing the solution, the following challenges were encountered:

- **Handling Optional "and"**:
    - In English, "and" is often included but not mandatory (e.g., "one hundred and twenty"). The parser had to be designed to ignore it safely.
- **Maintaining correct multiplication order**:
    - Words like "hundred" and "thousand" must **multiply only the immediate preceding number**, not the total so far, requiring careful pattern handling.
- **Recursion over lists**:
    - In Prolog, traditional iteration with counters (as used in imperative languages) is replaced by recursion with accumulators.
    - Designing correct recursive rules was critical to avoid infinite loops or incorrect parsing.
- **Debugging in Prolog**:
    - Debugging logic errors in Prolog differs from languages like Python — it requires careful tracing of rule applications and logical inference paths.

IV. Conclusions

The developed Prolog program successfully meets the requirements of **Variant 4** of the lab exercise.
It can accurately parse and convert English-written numbers (up to 1000) into their numeric forms.

Additionally:

- It correctly handles optional words,
- It deals with capitalization variations,
- It respects English grammar rules for number formation.

The lab offered valuable experience in **thinking declaratively** and applying **recursive logic processing** in Prolog — skills that differ significantly from procedural programming approaches used in languages like Python or Java.