Laboratory 7

Variant 3, Group 13

By Saumya Shah and Anas Tagui

Main Components & Logic

num_word

```
2   num_word(0, 'zero').
3   num_word(1, 'one').
4   num_word(2, 'two').
5   num_word(3, 'three').
6   num_word(4, 'four').
7   num_word(5, 'five').
8   num_word(6, 'six').
9   num_word(7, 'seven').
10  num_word(8, 'eight').
11  num_word(9, 'nine').
12  num_word(10, 'ten').
13  num_word(11, 'eleven').
14  num_word(12, 'twelve').
15  num_word(13, 'thirteen').
16  num_word(14, 'fourteen').
17  num_word(15, 'fifteen').
18  num_word(16, 'sixteen').
19  num_word(17, 'seventeen').
20  num_word(18, 'eighteen').
21  num_word(19, 'nineteen').
```

This predicate maps integers from 0 to 19 to their English word equivalents. It's used directly for numbers less than 20.

tens_word

```
24  tens_word(2, 'twenty').
25  tens_word(3, 'thirty').
26  tens_word(4, 'forty').
27  tens_word(5, 'fifty').
28  tens_word(6, 'sixty').
29  tens_word(7, 'seventy').
30  tens_word(8, 'eighty').
31  tens_word(9, 'ninety').
```

This maps the tens part of numbers (like 20, 30, ..., 90) to their word equivalents. It's used when dealing with numbers from 20 to 99.

to_words_lt100

```
% Helper predicate for numbers less than 100
to_words_lt100(N, WordsAtom) :-
    N < 20,
    num_word(N, WordsAtom).
to_words_lt100(N, WordsAtom) :-
    N >= 20,
    Tens is N // 10,
    Units is N mod 10,
    tens_word(Tens, TensWord),
    (   Units =:= 0 ->
        WordsAtom = TensWord
    ;   num_word(Units, UnitWord),
        atomic_list_concat([TensWord, UnitWord], ' ', WordsAtom)
    ).
```

A helper predicate to find the word representation for numbers less than 100. If the number is less than 20, it directly uses the `num_word` predicate. Otherwise, it calculates the tens digit (`Tens`) and the units digit (`Units`). Then, it retrieves the word for the tens digit using `tens_word(Tens, TensWord)`. If the `Units` digit is 0 (eg. for N=40), `WordsAtom` becomes just the `TensWord` (eg. 'forty'). Otherwise, it retrieves the word for the `Units` digit using `num_word(Units, UnitWord)`and then concatenates `TensWord` and `UnitWord` with a space in between to form `WordsAtom`.

`to_words_lt1000`

```
% Helper predicate for numbers less than 1000 (but >= 100)
to_words_lt1000(N, WordsAtom) :-
    Hundreds is N // 100,
    Remainder is N mod 100,
    num_word(Hundreds, HundredWord),
    (   Remainder =:= 0 ->
        atomic_list_concat([HundredWord, 'hundred'], ' ', WordsAtom)
    ;   to_words_lt100(Remainder, RemainderWords),
        atomic_list_concat([HundredWord, 'hundred', 'and', RemainderWords], ' ', WordsAtom)
    ).
```

A helper predicate to find the word representation for numbers less than 100. It calculates the hundreds digit (`Hundreds`) and the remainder (`Remainder`). First, it retrieves the word for the `Hundreds` digit using `num_words`, since it must be less than 10 (given our constraint `N <= 1000`). If the remainder is 0, we return the `HundredWord` concatenated with 'hundred'. Otherwise, we find the word representation of the remainder (which must be less than 100) with the `to_words_lt100` predicate, and then concatenate it as `HundredWord` + 'hundred and' `RemainderWord`.

`get_words_atom`

```
% Predicate to get the word atom (internal use)
get_words_atom(N, WordsAtom) :-
    integer(N), N >= 0, N =< 1000, % Input validation
    (   N =:= 0 ->
        num_word(0, WordsAtom)
    ;   N =:= 1000 ->
        WordsAtom = 'one thousand'
    ;   N < 100 ->
        to_words_lt100(N, WordsAtom)
    ;   N < 1000 -> % This implies N >= 100
        to_words_lt1000(N, WordsAtom)
    ).
```

This is the core predicate to convert numbers from 0 to 1000 into their word form:

- If `N = 0` → "zero"
- If `N = 1000` → "one thousand"
- If `N < 100` → uses `to_words_lt100`
- If `N < 1000` → uses `to_words_lt1000`

```
get_words_atom(N, 'Input out of range (0-1000)') :-
    \+ (integer(N), N >= 0, N =< 1000).
```

The other part of it acts as a fallback. If `N` is not an integer between 0 and 1000, it returns an error message.

`to_words`

```
% Main predicate to_words/1 (prints the words)
to_words(N) :-
    get_words_atom(N, WordsAtom),
    write(WordsAtom),
    nl.
```

This is the main predicate; it calls `get_words_atom` with the provided number and writes the output to the screen.

Challenges

- We were unfamiliar with Prolog's syntax and declarative paradigm, leading to initial learning difficulties.
- We also had to make sure we handled the special and edge cases properly (eg. numbers between 0 and 19, round numbers (20, 30, … 200, 300 …), one thousand, etcetera).