CS424/524 Programming Assignment #2

DUE (Section 02): April 13, 2015 by the end of the day.
DUE (Section 01): April 14, 2015 by the end of the day.

Very short description:

Write a program in Prolog (SWI-Prolog) that allows the user to encode a phrase using Morse code, decode a phrase encoded in Morse code, and convert each letter and digit in a phrase to its International Civil Aviation Organization (ICAO) phonetic alphabet equivalent.

More detailed description:

Your prolog program should allow the user to do three operations:

1. Convert each letter or digit in a phrase to its ICAO phonetic alphabet equivalent. For each letter and digit in the phrase, print the ICAO equivalent on a separate line for that letter or digit. Put a blank line between words. The user should be able to convert a phrase into ICAO using the following format:

   ```
   convertToIcao('Sentence to Convert').
   ```

   Note the result does not have to be stored and returned, just printed. For example:

   ```
   ?- convertToIcao('Star Wars').

   Sierra
   Tango
   Alfa
   Romeo

   Whiskey
   Alfa
   Romeo
   Sierra
   ```

2. Convert a phrase into Morse code. The user should be able to convert the phrase into Morse code using the following format:

   ```
   convertToMorse('Phrase to convert').
   ```

   The program must print the result of the conversion to the screen (note the result does not have to be stored and returned). A space should be placed between each encoded letter or digit. For any non-alphanumeric characters (other than spaces), the program should just copy the character to the output. Spaces should be replaced with a `/`, so that words will be separated with the `/`. For example:

   ```
   convertToMorse('Star Wars').

   ... - .- .-. / .-- .- .-. ...
   ```

3. Convert a phrase in Morse code back to its alphanumeric equivalent. The user should be able to decode the phrase using the following format:

   ```
   convertFromMorse('Encoded phrase to convert').
   ```

   Letters should be separated with a space and words should be separated with a `/`.
   For example:

   ```
   convertFromMorse('... - .- .-. / - .-. . -.-').

   STAR TREK
   ```

Note that if you have a fact

```
morse('O', '---').
```

That fact can be used to convert the `O` to `---` and to convert the `---` to `O`, depending on which item is bound. If you query

```
?- morse('O', Result).
```

Result will be `'---'`. If you query

```
?- morse(Result, '---').
```

Result will be `'O'`.

Here is a link to SWI-Prolog and a link to the documentation about predicates that operate on strings (which are very helpful for this program – contains predicates that allow you to split a string into characters and so forth). Also included is a Wikipedia link that lists ICAO and Morse code information.

http://www.swi-prolog.org

http://www.swi-prolog.org/pldoc/man?section=string-predicates

Remember that variables start with an uppercase letter in Prolog and constants start with lowercase.

Your facts and rules should be stored in a file named ***conversion.pl***.

Program Requirements:

Your program must do the following:

- Allow the user to convert each letter and digit in a phrase to its ICAO equivalent
- Allow the user to convert a phrase to Morse code
- Allow the user to convert a phrase from Morse code
- Include a comment block at the top of the program file that lists:
    - Your name
    - The date
    - The class and section (CS424-02 or CS524-02 or CS424-01)
    - An explanation of what the program does and what the format of the queries should be (how a user should format their queries)
- Use good identifier names (for rules, facts, variables, and constants) and appropriate comments throughout your program. Make use of whitespace (blank lines, etc.) in order to make your program readable

The program will be graded as follows:

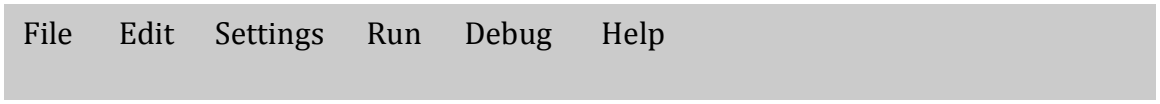
Correctly converts phrase to ICAO	25 points
Correctly converts phrase to Morse code	25 points
Correctly converts phrase from Morse code	25 points
Readability of program	10 points
Other (anything else that affects the usability or effectiveness of your program or relates to the specified requirements)	15 points

The program should be named ***conversion.pl*** and the .pl file should be submitted through Canvas.

Note: The next page contains a description for using SWI-Prolog, from Dr. Weisskopf's past CS424/524 class.

RUNNING A PROLOG PROGRAM IN THE CS DEPARTMENT

1. Create the program as a text file and save with a .pl extension, e.g. `myProg.pl`
2. From the Start Menu select SWI-Prolog/Prolog. The following window will open:




```
% library(win_menu) compiled into win_menu 0.02 sec, 11,720 bytes
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 2,024 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.2)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).
```

1. To run the program, first select File/Consult. Navigate to the directory where the file is stored, select the file, and double-click Open. You will see a message similar to the following:
   ```
   % c:/Temp/owner.pl compiled 0.00 sec, 2,164 bytes
   ```
   If there are compiler errors they will be displayed in a window.

2. Now you are ready to type in queries. Be sure to end every query with a period ('.'). Prolog will "hang" until you do so. The Prolog prompt for input is *?-*

3. For example, if you have loaded and compiled the "owner" program, try typing in
   ```
   1 ?- dog(spot).
   ```
   Prolog will respond
   ```
   true.
   ```

4. If you want to edit the file before or after compiling, select File/Edit from the drop-down menu. Navigate to the directory where your file is stored; select the file and then click Open. A copy of the file will open. You can modify it as you desire. Be sure to save the file after making any changes. Use the save-buffer option on the File drop-down menu.

5. To run the program after modifications, you can select File/Reload modified file which will load and compile the file, or you can go back to step 3 and select File/Consult.

6. Experiment with the file; for example, to see all cats: enter cat(X).
   ```
   5 ?- cat(X).      % don't forget the period

   X = fluffy
   ```

7. Now to see if there are other solutions, hit the semicolon key
   ```
   X = fluffy ;
   X = patrice.

   6 ?-  There are no other possibilities.
   ```

8. At any time you may go back and modify the program file, either by changing or adding rules or by adding and deleting facts. Before you try to run the program again be sure to save the file and re-compile Some other options you may find useful: Under the Run option on the menu bar, Interrupt may stop a runaway program. New thread will start a new instance of the program.

3. OR….navigate directly to the .pl program and double click on the icon. The window described in step 2 above will open, your program will be compiled, and you can begin to test your program. All other instructions above about modifying, recompling, etc. will still work