Question 1: Summaries

Now, we are going to change the functionality that you just added for summarizing the slot values. For example, we might want the system to say “turning on light in kitchen.”. Let the dialogue designer specify a summary using the global variable `summary`, which the designer can specify with variables for the slot name, using angled brackets. So, for this application, the summary could be

```plaintext
set summary {turning <state> <device> in <room>.}
```

Change your code so that once the slot is finished, you fill in the appropriate slot values, and you pass the message to the TTS engine.

Writeup 1.1. Marks 2 Hand-in your code for Converse.

Question 2: Help

Add functionality to `GetResponse` so that the user can say ‘help’. In this case, the system should automatically list all of the allowed parameter values.


Hierarchical Forms

Change your dialogue manager so that it can deal with hierarchical forms. Below is an example hierarchical form for ordering food. The top level form specifies that the value for the hamburger slot can be yes or no, that drink can be yes or no, and that fries can be yes or no. If hamburger is yes, the system should prompt for whether it should have cheese on it, and if it should have bacon on it. If drink is yes, you need to find out the size, and what type of soda. If fries is yes, whether they should be curly fries or not.

```plaintext
set topform foodorder
set formslots(foodorder) {hamburger drink fries}
set prompt(hamburger) {do you want a hamburger?}
set grammar(hamburger) {yes|hamburgerorder | no}
set formslots(hamburgerorder) {cheese bacon}
set prompt(cheese) {do you want cheese on your hamburger?}
set grammar(cheese) {yes | no}
set prompt(bacon) {do you want bacon on your hamburger?}
set grammar(bacon) {yes | no}
set prompt(drink) {do you want a drink?}
set grammar(drink) {yes|drinkorder | no}
set formslots(drinkorder) {drinktype drinksise}
set prompt(drinktype) {what type of drink do you want?}
set grammar(drinktype) {coke | orange | root beer | lemonade}
set prompt(drinksise) {what size of drink do you want?}
```
set grammar(drinks) {small | medium | large}

set prompt(fries) {do you want french fries?}
set grammar(fries) {yes | no}
set formslots(fries) {friestype}
set prompt(friestype) {what kind of fries do you want?}
set grammar(friestype) {plain | curly}

To make this work, you should keep a list (or agenda) or slots that you still need to work on. As you fill in each one, remove it from the list. When you encounter a hierarchical form, you can simply add its slot fillers to the list.

Your dialogue manager should also be able to hand ‘repeat’, ‘start over’, and ‘quit’.

Writeup 2.3. Marks 10
Hand in a copy of the relevant code. Also hand-in several test runs (text transcript of what system said and system thought user respond back with) that show the features of your system.

Question 3: Used Car Application

Using your dialogue manager from Question 2, build an application that helps a user find a used car from a database.

To help you along with this, I created a database of 2000 cars that differ along the dimensions given below.

<table>
<thead>
<tr>
<th>color</th>
<th>black</th>
<th>white</th>
<th>red</th>
<th>blue</th>
<th>brown</th>
<th>green</th>
<th>yellow</th>
<th>purple</th>
<th>grey</th>
</tr>
</thead>
<tbody>
<tr>
<td>doors</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>powerwindows</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>powerbrakes</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>powersteering</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transmission</td>
<td>manual</td>
<td>automatic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cylinders</td>
<td>4 6 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>van car suv truck</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>airbags</td>
<td>yes no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mileage</td>
<td>10000 20000 30000 40000 50000 60000 70000 80000 90000 100000 110000 120000 130000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The list of cars is in the file hw5cars.tcl, available from the course webpage, right by the link to this homework assignment. To read in the car database, just use the tcl source command in your program.

source hw5cars.tcl

To further help you, use the following routine findcars, which returns a list of cars that satisfy the constraints met by the values specified in the associative array called ::values.

proc findcars {} {
    set cars {}
    for {set c 0} {$c < $::numcars} {incr c} {
        set found 1
        foreach {slot val} [set ::car$c] {
            if {![[info exists ::value($slot)]]} continue
            if {$::value($slot) != $val} {
                set found 0
                break
            }
        }
        if $found
            lappend cars $::car$c
    }
    return $cars
}
Here is some code to illustrate how findcars works.

```bash
set ::value(color) black
set ::value(cylinders) 4
set ::value(year) 2005
puts [findcars]
```

Get a minimum version of the system working.

**Writeup 3.4. Marks 3**  Hand in the code that you changed, and hand in two actual dialogues that a user can have with the system.

Below is a list of simple to more sophisticated versions you can build.

1. Allow user to specify “doesn’t matter” for any slot
2. Give the user feedback as to how many cars match the slots specified so far. If there is only one car, do not ask any further slots.
3. Allow the user to undo a slot-filler pair if no cars matches the slots.
4. Once the number reaches 5 cars, describe each car to the user.
5. Let the user interrupt the system when the system is listing all of the cars that are left.
6. Pick the order that the slots are asked to minimize the expected length of the conversation or to minimize speech recognition failure.

Add extra functionality to it to make the system work better.

**Writeup 3.5. Marks 3**  Describe the functionality you added. Hand in your code, and highlight what code you changed to add in the functionality.