CSE 341 Programming Assignment 3: Prolog

Due: March 13

Prolog is good at pattern matching, parsing, and interpreting. It also has that nice built-in search mechanism that can be used for reasoning. Thus it is an ideal language in which to write a program that "understands" a natural language, in the sense that it can parse sentences, interpret them, and respond. That is the goal of this assignment.

Your program should recognize English sentences having the following forms:

1. <name> is [a | an] <object class>.
2. [a | an] <object class1> is [a | an] <object class2>.
3. is <name> [a | an] <object class>?
4. [a | an] <object class> has <number> <part name>[s].
5. how many <part name>s does <name> have?

Here are some example sentences.

```
john is a man.
a man is a person.
is john a man?
is john a person?
is mary a person?
mary is a woman.
a woman is a person.
is mary a person?
a person has 10 fingers.
how many fingers does mary have?
```

But what does "recognize" mean? First, it means to parse the sentence, recognizing which of the forms it is and saving the names, numbers, and object classes in appropriate variables. The second thing it means is to *take some action* for that sentence. For declarative sentences, the action is to assert a fact or rule into the database. For questions, the action is to construct a call to query the database according to the question. For each of the above sentence forms, here is what has to be constructed and then either asserted or called.

1. assert <object class>(<name>).
2. assert <object class2>(X) :- <object class1>(X).
3. call <object class>(<name>).
4. assert <has(<object class>,<number>,<part name>[s])
5. call <has(<name>,<variable>,<part name>[s])

This is what each of the example sentences should cause to happen.

```
assert man(john).
assert person(X) :- man(X).
call   man(john).
call   person(john).
call   person(mary).
assert woman(mary).
assert person(X) :- woman(X).
call   person(mary).
assert has(person,10,fingers).
call   has(mary,X,fingers).
```

There is a piece of knowledge missing from the above (for reasons of simplifying the assignment) that is needed in the reasoning. This knowledge is that if a particular object class has N parts of a particular type and if a certain entity X belongs to that object class, then that particular entity also has N parts of that type. You may write static rules to take care of this knowledge, or, for EXTRA CREDIT, you can modify the action for sentence form 4. to dynamically assert this kind of rule for every object class and part name that the user mentions. For instance, if the user says,

```
a horse has 4 legs.
```

then, **has(X,N,legs) :- horse(X), has(horse,N,legs)** would be asserted. Or, perhaps you can figure out an even more clever way of doing this.

In addition to performing the asserts and calls, your program should give the user some feedback on each sentence he/she types in. For declarative sentences, you should print “ok” and go to a new line. For questions, you should print an appropriate answer to the question. For example,

```
[john,is,a,man].
ok
[a,man,is,a,person].
ok
[is,john,a,person,?].
yes
[is,mary,a,person,?]
unknown
[a,person,has,2,legs].
ok
[how,many,legs,does,john,have].
2
```

Notice that the sentences are input as lists. Thus your pattern matching should match against lists. This is the easiest way to do it; you're welcome to take it further, but you don't have to.

You should have a main rule that reads a sentence list, parses it, and responds to it, in a loop. You can use *repeat* to loop.

```
talk :- repeat, <loop body>.
```

You will receive further instructions on what to test it on and what to turn in.