Programming Assignment #2

CS 6353 Spring 2015

In this programming assignment, you will be using the Rust programming language to model simple serialized data structures that may reference each other, and then will be answering queries about those data structures. The requirements for your submission are listed in sections 1 and 2 below. *Follow the requirements carefully and precisely – they will be thoroughly tested.* Section 3 provides a few example outputs.

I will grade your program by running it against a large test suite. You will be graded by the percentage of test cases you get correct.

The Due Date for this assignment is April 8th at 11:55pm.

1. Program Specification

Your program will read from standard input and output to standard output.

Spaces and tabs can both be used to separate tokens. Multiple spaces and tabs, or any combination thereof are allowed.
In the below specification, non-terminals are capitalized. Terminals are either in regular expression syntax, or single quotes if they are literal.
Here is a specification for valid inputs:

```
INPUT   ::= DATA '.' '\n' QUERY QUIT;  # \n means enter is hit.
QUIT    ::= 'QUIT' '\n';
DATA    ::= LIST '\n' DATA | ;   # Note: "| " means match the empty string.
LIST    ::= '{' NAME ':' ITEMS '}';
NAME    ::= STRING;
ITEMS   ::= ONEITEM | ONEITEM ',' ITEMS;
ONEITEM ::= NUMBER | STRING | PTR | LIST;
NUMBER  ::= '-' [0-9]+ | [0-9]+;
STRING  ::= [a-zA-Z][0-9a-zA-Z]*;
PTR     ::= '@' STRING;
QUERY   ::= ONEQ '\n' QUERY | ;
ONEQ    ::= 'SUM' | 'sum' | 'NAMECHECK' 'namecheck' |
            'PTRS' | 'ptrs' | 'SEARCH' STRING |
            'search' STRING | NUMBER;
```

Your program should read in ALL my input, then make a determination if the input is SYNTACTICALLY well-formed or not. If **any** of the input is ill-formed, then your program should only print ERR, then a new line, and then exit.

Otherwise, your program should produce an answer for every query line, on a line by itself.

The semantics for a SUM query: You are to calculate the total of any numeric "ITEM" field in the given data set, counting each field once and only once.

The syntax for SUM output is:

```
SUMOUT ::= NUMBER '\n';
```

The semantics of a NAMECHECK query: You are to determine whether or not each NAME field is unique. If there are no conflicts, you will print 'OK' (see output spec below, as well as examples), otherwise you will output a list of names that conflict, sorted in alphabetical order (outputting each name only once). Note that the name field is CASE SENSITIVE.

The syntax for NAMECHECK output is:

```
NAMECHECKOUT ::= 'OK' '\n' | NAMELIST '\n';
NAMELIST     ::= STRING | STRING ',' NAMELIST;
```

The semantics of a PTRS query: The @ followed by a string represents a "pointer" to a list, referring to the name of the list. The PTRS query should print 'OK' if there are no pointers, or if all pointers reference a valid list. Otherwise, the command should output the missing target of each bad (aka 'dangling') pointer, (without the @), sorted in alphabetical order, and outputting each only once. Targets are case sensitive.

The syntax for PTRS output is identical to the syntax for NAMECHECK output.

The semantics of a SEARCH query: For each list that contains either the exact, case sensitive search string or number to be search as a match: print the name of the list, along with nested list names. For instance, if we search for "foo", and find it only in a list named "x", and the list named x is defined in a list named "y", and the list "y" is defined in a list named "z", you will end up outputting x:y:z. Note that this command does not "follow pointers", it simply looks at how lists are declared. If the search term is not found in any item, then you will output NIL.
Sort the SEARCH output in alphabetical order.

The syntax for SEARCH output is:

```
SEARCHOUT ::= MATCH '\n' | MATCH ',' SEARCHOUT '\n' | 'NIL' '\n';
MATCH     ::= STRING | STRING ':' MATCH;
```

Note that each query I make in my testing will count as a separate test case when grading. It helps to get the grammar correct—if your program wrongly thinks that an input is invalid but I actually launch 10 queries, you will lose out on 10 cases!

2. Other Requirements

You will receive a 0 on this if any of these requirements are not met!

1. The program must be written entirely in Rust, with no third party libraries (just ones that ship with the language).
2. You must use a single input file NETID.rs as the name. For instance, for me, it would be jtv202.rs. Note, I will compile your program using rustc NETID.rs
3. The program must compile and run in the reference environment. Even if it works on your desktop, if it doesn't work in the reference environment, you will get a 0.
4. You must submit your homework before 11:55pm on the due date. You will lose 15 points for each day you are late.
5. You must submit the homework through the course website, unless otherwise pre-approved by the professor.
6. You may not give or receive any help from other people on this assignment, except the professor or TA.

7. You may use references on the Internet to teach yourself Rust.
8. Again, you may NOT use code from any other program, no matter who authored it. This includes 3rd party libraries (you may use anything that comes with Rust by default).
9. As a warning, I generally catch people who collaborate with other students, who use web sites on the internet to beg or pay other people to write their code for them, etc. If I catch you, you will receive a 0 on the assignment and get reported to the department.

3. Test Cases

Below are four sample test cases for you, which I will use in my testing. Typically, I use anywhere from 20-40 test cases, and will definitely use these three. I strongly recommend you create your own test harness and come up with a large number of test cases to help you get the best possible grade.

For test cases, what one would type on the command line is **BLACK**, input is in **GREEN**, and output is in **BLUE**.

Case 1

```
./NETID
{a : 1, 2, hello, {b: 100, @c}}
{c : goodbye}
.
SUM
NAMECHECK
PTRS
SEARCH 100
QUIT
103
OK
OK
b:a
```

Case 2

```
./NETID
{a : 1, 2, hello, {a: 100, @x}}
{b : 1, 2, hello, {b : 100, @y}}
.
SEARCH 100
NAMECHECK
PTRS
QUIT
a:a,b:b
a,b
x,y
```

Case 3

```
./NETID
{a : 1, 2, 3}
SUM
QUIT
ERR
```

Note that, for the third output, there is a missing period, otherwise we would expect 6 as output.

4. Reference Environment

We are using the same reference environment as before.

5. Resources

The main Rust tutorial is here: https://doc.rust-lang.org/book/
Other documentation can be found off the main Rust web site: http://www.rust-lang.org/