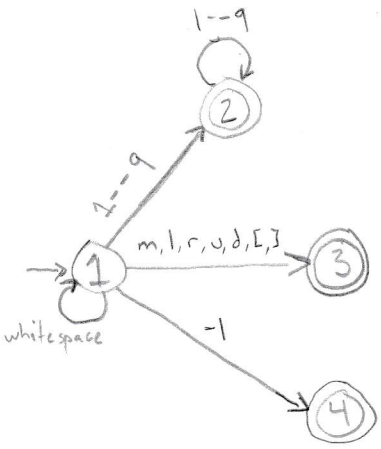
Code for Problem 1 goes here:

```
do {
   sym = getNextSymbol();

   if ( state == 1 ) {
      if ( sym == '?' ) {
         data += (char) sym;

         state = 2;

      }
      else if ( letter(sym) ) {
          data += (char) sym;

          state = 3;

      }
   }
   else if (state == 2) {
          if(letter(sym)){
            data += (char) sym;
            state = 2;
          }
          else {
            putBackSymbol(sym);
            done = true;
          }

   }
   else if (state == 3){
       if (digit(sym)){
          data += (char) sym;
          state = 4;
       }
       else{
          putBackSymbol(sym);
          done = true;
       }

   }
   else if (state == 4) {
        if (digit(sym)){
           data += (char) sym;
           state = 3;
        }
        else {
           putBackSymbol(sym);
           done = true;
        }
   }
} while(!done);
```

2. At the course web site, in the folder `Spring2020/3210s20/Test2`, you will find a file `Lexer.java` that implements the lexical part of the `TurtleLang` language.

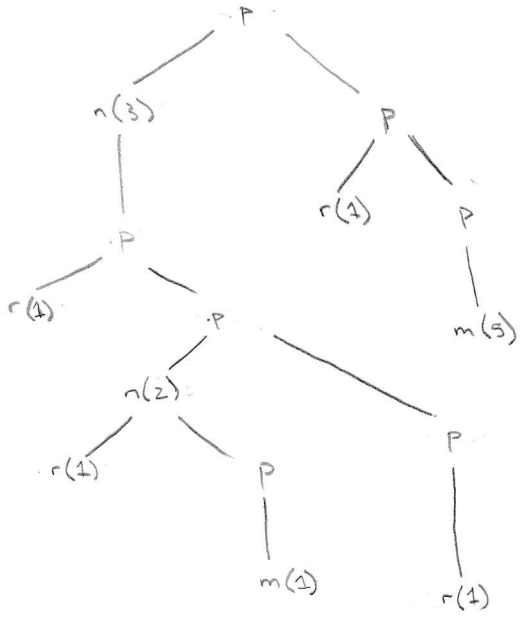
   Draw below the finite automaton implemented in that code.

   Be sure to clearly indicate the start state by drawing an arrow from no state to it, indicate accepting states by double-circling them, and label every arc with either a single symbol, a range of symbols such as `a--z`, or a list of symbols separated by commas like `a, e, i, o, u`.



3. Draw below the parse tree that should be produced by the parser for the input string
`3[r2[rm]r]r5m`

You may abbreviate the grammar symbols as has been done in the tree shown previously.



4. In the folder for this test you will find the file `Parser.java`.

Your job on this problem is to write (either by hand below or in the file, but attach a photo of your code either way) the body of the `parsePath` method.

```
public Node parsePath() {
   System.out.println("-----> parsing <path>:");
   // insert your code for Problem 4 here:
   Node first = parseAction();
   Token token = lex.getNextToken();
   if (token.matches("]", "")) {
     lex.putBackToken(token);
     return new Node("path", first, null, null);
   }
   else if (token.isKind("eof")) {
     return new Node("path", first, null, null);
   }
   else {
     lex.putBackToken(token);
     Node second = parsePath();
     return new Node("path", first, second, null);
   }
}// parsePath
```

5. In the folder for this test you will find the file `Node.java`.

   Your job on this problem is to write (either by hand below or in the file, but attach a photo of your code either way) the missing cases in the `execute` method for the kinds `path` and `num`.

```
if ( kind.equals("path") ) {
   // insert your first chunk of code for Problem 5 here:
   first.execute();
   if (second != null) {
      second.execute();
   }
```

```
}// path

else if ( kind.equals("num") ) {
   // insert your second chunk of code for Problem 5 here:
   int num = Integer.parseInt(number);
   for (int k = 1; k <= num; k++) {
      first.execute();
   }
```

```
}// num
```