Comp 333 Project #2 (Recursive Descent Parser Project)

Due: Tuesday Feb 24, 2015 (See below)
Points: 20

1. Study the attached RecDescentParser.java program. Notice that the input program is a string of characters. Run it with a few test cases to see how it works. Your test cases should be programs that are accepted and programs that are not accepted. Note that this program accepts strings satisfying the grammar
   S → E$
   E → T + E | T
   T → F * T | F
   F → (E) | *letter* | *digit*

In parts 2 and 3 of this assignment you will be modifying the parser to accept additional grammar rules. Do not make ANY changes to the constructor or to the getChar(), match(), error() or start() methods. They need to be as they are for instructor testing purposes.

2. Modify the RecDescentParser.java program to accept strings satisfying
   S → N = E$
   E → T + E | T – E | T
   T → F * T | F / T | F
   F → (E) | *letter* | *digit*
   N → *letter*

3. Next modify the program from Step 2 to accept strings satisfying
   S → STList $
   STList → ST | ST ; STList
   ST → N = E
   E → T + E | T – E | T
   T → F * T | F / T | F
   F → (E) | *letter* | I
   N → *letter*
   I → *digit* | *digit* I

4. When you are completed with Steps 2 and 3 you should have a **single** program in a file called RecDescentParser.java. That is the program that will be submitted to Moodle and tested by your instructor.

5. Test your program thoroughly. Test all aspects of the grammar. Test good and bad strings. You do not need to turn in these test cases. Your instructor will run your program on a set of test cases which will not be known to you. At least 50% of your grade will be based on how well your program runs on the instructor test cases. [Suggestion: appoint someone on your team to thoroughly test your program.]

6. **Submit your RecDescentParser.java program to Moodle on Feb 24 by 7:00 am**. All team members must be listed in the program. Your program must be named RecDescentParser.java and must have a public start() method, otherwise your program cannot be tested.

7. **Hand in in class on Feb 24: (stapled together)**
    a. Title Page with course , project, date and team member names.
    b. RecDescentParser.java source code with the printout of following 4 test cases commented out at the bottom of the Java file.

        " x = y * (a + 2 – c) + A $"
        "z = a + (b + 1)) $"
        " u = 7 ; v = a + 8 ; x = 3$"
        "c = z + 341 + 5 ; x = 100$"

```
/*Recursive descent parser for expressions

  To simplify the programming tokens will be single characters
  consisting of  digits, letters, *, + ,(, ) ,  =,  $
  Input strings will look like :

  "x + a * ( 9 + b)$"
  White space is ignored.

   S -> E$
     E -> T + E  | T
     T -> F * T  | F
     F -> (E) | letters | digits

*/

  import java.util.*;

  public class RecDescentParser
  {
      public char nextchar;
      public String inputString;
      private int pos;

      public RecDescentParser ( String s)
      {
         inputString = s;
         pos= 0;
         System.out.println("\nParsing:  " + s);
      }

      public void start()throws Exception
      {
         try {
            getChar();
            S();
            System.out.println("Successful parse");
         }
         catch (Exception e)
         {
            System.out.println("Unsuccessful parse");
         }

      }

      public void getChar()throws Exception
      {
         //skip over blank chars
         while ( pos < inputString.length()
               && inputString.charAt(pos) == ' ')
         {

               pos++;
         }

         if( pos < inputString.length() )
         {
           nextchar = inputString.charAt(pos);
           System.out.println("getChar:  " + pos + "  "  + nextchar);
           pos++;
     }
     else
          error();

  }

  public void S()throws Exception
  {
     E();
     match('$');
  }


     public void E()throws Exception
  {

        T();
        if( nextchar == '+')
        {
           match('+');
           E();
        }

  }

  public void T()throws Exception
  {

        F();
        if( nextchar == '*')
        {
           match('*');
          T();
        }
 }


  public void F()throws Exception
  {

    if ( Character.isLetterOrDigit(nextchar))
     {
        match(nextchar);
        return;
     }
     else if (nextchar == '(')
     {
        match('(');
        E();
        match(')');
     }
     else
        error();
  }
```

```
    public void error() throws Exception
    {
       System.out.println( "Syntax error at position : "
                + pos  +  "  with character: " + nextchar);
       throw new Exception ("Syntax Error");
    }

   public void match( char u) throws Exception
    {
       if(nextchar == u)
       {
          if ( u != '$')
             getChar();
          return ;
       }
       else
         error();
    }


    public static void main( String[] args)throws Exception
    {

          RecDescentParser rdp =
                      new RecDescentParser("x * y + 5  * (9 + y) $"  );
          rdp.start();


          RecDescentParser rdp2 =
                      new RecDescentParser("x * y (+ 5 ) * (9 + y) $"  );
          rdp2.start();

           RecDescentParser rdp3 =
                       new RecDescentParser("x * y + 5  * (9 + y) $"  );
          rdp3.start();

          RecDescentParser rdp4 =
                       new RecDescentParser("x * 8 "  );
          rdp4.start();


    }

}
```