Comp 333 Project #4 (Java to Scheme Conversion Project)

**Due: April 21**
**Points: 20**

**This is an individual or a team project ( Your choice). Teams may have up to three members. Each team should submit only one project with all team member names the uploaded file. Individuals submit individual projects.**

1. Study the attached Recursive Descent Parser1 program. This program is very similar to the recursive descent parser in Project 2. It parses strings of letters, digits, operators, $ and spaces using the grammar
   S → E$
   E → T + E | T – E | T
   T → F * T | F / T | F
   F → ( E ) | *letters* | *digits*

2. Translate the RecursiveDescentParser1 program to a Scheme program. Use Dr Racket. The translation should be a "fairly close" translation. Use the same names for the functions. There will of course be some needed changes due to the differences in the languages. Follow these guidelines.

   a. Define nextchar, pos and tokenList as global( top level) variables.

   b. Create a function called parse which takes as an argument the string to be parsed and immediately converts it to a list of characters with no spaces. The list of characters is then stored in tokenList . Do this as follows:

```
(define parse
  (lambda  (inputstring)
    (set!    tokenList   (removeSpaces ( string->list     inputstring)  ) )
     ;display the tokenlist
     ( start)  )  )
```

   c. You will need to write the function removeSpaces that removes all spaces from a list of characters. Check Dr Racket to see what string->list does.

   d. Now write the functions start, getchar, match, S, E, T, F . These functions should access and modify the variables nextchar, pos and tokenList. The getchar function should printout the same information as the java getchar() with the

exception that pos is the position of the token in the tokenList, not the position of the character input string.

e. Ignore the Exception mechanisms in the Java program. Instead replace all calls to the error() function with a call to the following Scheme function.

```
(define syntaxError
  ( lambda ( )
    (raise-user-error
        "Syntax error at token position: "     (-    pos    1)
         "with next character: "     nextchar )     ) )
```

f. The *raise-user-error* Scheme function prints the arguments and then stops the program. This is one way that Scheme raises and handles exceptions.

g. You may add other helper functions if needed.

3. Test Cases
   a. (parse " x + y – 5$" )
   b. (parse " 3 * ( q + w / 5) $")
   c. (parse " ( a * ( b + c/ ( 3 * d) ) )" )
   d. (parse "5 + 6 * 8) $" )
   e. (parse " 1 * 2 ) - 3 + z $" )

.

4. **Hard copy submission. Due in Class on April 21**
   a. Submit title page, source code, test cases with test case runs.
5. **Moodle submission: Submit to Moodle by April 21 at 7 am**
   a. ONE file with all of your Scheme functions plus your commented out testcases with results from Item 4. Name the file recursiveDescentParser.rkt
   b. Make sure all team member names are embedded in the file.
   c. The instructor will be testing this file and so it needs to be complete. Missing functions means it won't pass the instructors' tests.

```
/*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 - E  | T
     T -> F * T  | F / T  | F
     F -> ( E ) | letters | digits


*/

  import java.util.*;

  public class RecDescentParser1
  {

      public char nextchar;
      public String inputString;
      private int pos;

      public RecDescentParser1 ( 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();
      }
      else if ( nextchar == '-')
      {
         match('-');
         E();
      }

}

public void T()throws Exception
{

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

           RecDescentParser1 rdp =
                new RecDescentParser1("x * y + 5  * (9 / y) $"  );
           rdp.start();


           RecDescentParser1 rdp2 =
                 new RecDescentParser1("x * y (+ 5 ) * (9 + y) $"  );
           rdp2.start();

            RecDescentParser1 rdp3 =
                  new RecDescentParser1("x * y - 5  * (9 + y) $"  );
           rdp3.start();

           RecDescentParser1 rdp4 =
                   new RecDescentParser1("x * 8 "  );
           rdp4.start();

     }

}
```