UNIT 7 ASSIGNMENT 1

Name :Spuritha Mudireddy

CSULB ID: 030743269

In this assignment we are to do an extended version Unit 3 Assignment 2 WITHOUT the benefit of the ANTLR generated parser and lexer.

CODE:

```
import java.util.*;
class Expression {
   public static  HashMap<String,Integer> memory=new HashMap<String,Integer>();
   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);
      String in;
   /*Scanning the input line by line and parsing it*/
      while(sc.hasNextLine())
      {
         in=sc.nextLine();
         parse(in);
      }

   }

   public static void parse(String in)
   {

      String prt[]=in.split(" ");
      for(int i=0;i<prt.length;i++)
      {
         /*
         Similar to this grammary rule
         let : 'LET' ID '=' expr {memory.put($ID.text, new Integer($expr.value));};
         The variable and expression values are stored in a hashmap 'memory'
          */
         if(prt[i].equals("LET"))
         {
```

```
            memory.put(toString(prt[i+1].charAt(0)),expr(prt[i+1].substring(2)));

            for (String name: memory.keySet()) {
                String key = name.toString();
                String value = memory.get(name).toString();
            }
        }


        else if(prt[i].equals("PRINTLN"))
        {

            System.out.println(atom(prt[i+1]));
        }

        else if(prt[i].equals("PRINT"))
        {

            System.out.print(atom(prt[i+1]));
        }
    }
}


/*expr() is to evaluate the expression*/
public static int expr(String s)
{

    Stack<Integer> v=new Stack<Integer>();
    Stack<String> op=new Stack<String>();
    int i=0;
    while(i<s.length())
    {
        String p=toString(s.charAt(i));
        /*If it is a number ,the entire number is pushed into the stack 'v'*/
        if(isNumeric(p))
        {
            String num="";
            int in=i;
            while(in<s.length()&&isNumeric(s.substring(in,in+1)))
            {

                num+=s.charAt(in);
```

```
            in++;
        }
        i=in-1;
        v.push(Integer.parseInt(num));


    }

    /*If it is a variable , then its value from hashmap is pushed into the 'v' stack*/
    else if(memory.containsKey(p))
    {

        v.push(memory.get(p));

    }

    /*If it is a '(' ,then the inner expression between '()' is evaluated by making a recursive call and
then the value is pushed into the stack 'v'*/
    else if(s.charAt(i)=='(')
    {
        String brac="";
        int x=i+1;
        while(x<s.length()&&s.charAt(x)!=')')
        {
            brac+=s.charAt(x);

            x++;
        }
        i=x;

        if(isNumeric(brac))
            v.push(Integer.parseInt(brac));
        else
            v.push(expr(brac));

    }

    /*If it is operator , then the precedence is checked with the previous operator and the  value is
calculated*/
    else  if(isOperator(p))
    {

        while(op.size()>0&&precedence(op.peek(),p))
        {
```

```
            String c=op.pop();
            int op1=v.pop();
            int op2=v.pop();
            v.push(calculate(op1,op2,c));
         }
         op.push(p);

      }
      i++;
   }
   while(op.size()>0)
   {
      String c=op.pop();
      int op1=v.pop();
      int op2=v.pop();
      v.push(calculate(op1,op2,c));
   }

   return v.pop();

}


public static int atom(String s)
{

   if(s.length()==1)
   {
      if(isNumeric(toString(s.charAt(0))))
         return Integer.parseInt(s);
      else
         return memory.get(s);

   }

   else
      return expr(s);

}

public static String toString(char ch)
{
   return Character.toString(ch);
```

```
}
/*To check if a string is a number*/
public static  boolean isNumeric(String s)
{

   try
   {
      Integer.parseInt(s);
      return true;
   }
   catch( Exception e )
   {
      return false;
   }
}

/*To check if a string is an operator*/
public static boolean isOperator(String s)
{
   if(s.equals("+")||s.equals("-")||s.equals("*")||s.equals("/"))
      return true;
   return false;
}

/*To calculate an expression*/
public static int calculate(int op2,int op1,String op)
{

   if(op.equals("+"))
      return op1+op2;
   else if(op.equals("-"))
      return op1-op2;
   else if(op.equals("*"))
      return op1*op2;
   else
      return op1/op2;
}

/*To check the precedence of the operators*/
public static boolean precedence(String op1,String op2)
{
   HashMap<String,Integer> hm=new HashMap<String,Integer>();
   hm.put("+",1);
   hm.put("-",1);
```

```
        hm.put("*",2);
        hm.put("/",2);

        if(hm.get(op1)-hm.get(op2)>0)
            return true;
        else
            return false;

    }

}
```

Output: