//Nicholas Williams

import java.io.\*;

import java.util.\*;

public class Parser {

private static boolean debugging = true;

private static String outputFile = "ParserOutput.txt";

private static String[] Token = new String[2];

private static Stack<String> stack = new Stack<String>();

private static String currentStack;

public static boolean error = false;

private static int debugStep;

private static Integer operation;

private static boolean currentToken = true;

private static String[] columnArray;

private static String[] rowArray;

private static String[] grammar;

private static int rowCount;

private static Hashtable <String, Integer> parse = new Hashtable<String, Integer>();

//converts a string to all upercase letters.

private static String UpperCase (String input){

String temp = "";

for (int i = 0; i<input.length(); i++){

temp = temp + Character.toUpperCase(input.charAt(i));

}

input = temp;

return input;

}

//loads a parse table from a file name

private static void getParseTable(String fileName)throws FileNotFoundException, IOException{

File file = new File(fileName);

String st;

int stringIndex = 0;

BufferedReader br = new BufferedReader(new FileReader(file));

st = br.readLine();

columnArray = st.split(",");

rowCount = 1;

while((st = br.readLine()) != null){

rowCount++;

}

rowArray = new String[rowCount];

rowArray[0] = columnArray[0];

//parseTable = new String[rowArray.length-1][columnArray.length-1];

br.close();

br = new BufferedReader(new FileReader(file));

rowCount=0;

st = br.readLine();

while((st = br.readLine()) != null){

rowCount++;

stringIndex = 0;

String[] storage = st.split(",");

storage[stringIndex] = UpperCase(storage[stringIndex]);

rowArray[rowCount] = storage[stringIndex];

stringIndex++;

while(stringIndex<storage.length)

{

if(storage[stringIndex].equals("999"))

stringIndex++;

else{

parse.put((rowArray[rowCount] + columnArray[stringIndex]), Integer.parseInt(storage[stringIndex]));

stringIndex++;

}

}

}

br.close();

}

//loads the grammar from file name give in main;

private static void loadGrammar(String fileName)throws FileNotFoundException, IOException{

File file = new File(fileName);

String st;

String currentString = "";

BufferedReader br = new BufferedReader(new FileReader(file));

while((st = br.readLine()) != null){

currentString = currentString + "~" + st;

}

grammar = currentString.split("~");

br.close();

}

//when called from the debugger, will print the stack;

public static void DumpStack(){

System.out.println(">>- " + debugStep +" -<<");

System.out.println("Stack ::==> " + stack);

System.out.print("Popped " + currentStack + " with token " + Token[0]);

}

//does the operation as explained in the grammar based on the value from the parse table

public static void grammarOperations(){

if (operation < 0){

if (debugging){

DumpStack();

System.out.println(" -> $ PUSH $ @ EPSILON @ " + currentStack + " ::= @ EPSILON @");

System.out.println();

}

}

else{

String[] x = grammar[operation].split("::= ");

if(x.length > 1){

String[] commands = x[1].split(" ");

if (debugging){

DumpStack();

System.out.println(" -> $ PUSH $ [ " + operation +" ] " + currentStack + " ::= [" + x[1] + "]");

System.out.println();

}

for(int i = 1; i <= commands.length; i++){

if((!(commands[commands.length-i].equals(""))) && (!(commands[commands.length-i].equals(" "))))

stack.push(commands[commands.length - i]);

}

}

}

}

public static void main(String args[]) throws FileNotFoundException, IOException

{

SymbolTable.createSymbolTable("Global");

SymbolTable.createSymbolTable("Constant");

SymbolTable.insert(SymbolTable.Global, new ProcedureEntry("READ", 0));

SymbolTable.insert(SymbolTable.Global, new ProcedureEntry("Write", 0));

SymbolTable.insert(SymbolTable.Global, new VariableEntry("INPUT",0,""));

SymbolTable.insert(SymbolTable.Global, new VariableEntry("OUTPUT",0,""));

if(debugging){

PrintStream o = new PrintStream(new File(outputFile));

PrintStream console = System.out;

System.setOut(o);

}

Lexer.load("test.txt");

loadGrammar("vascalGrammar.txt");

getParseTable("parseTable.txt");

Token[0] = "";

Token[1] = "";

String holder;

debugStep = 0;

stack.push("ENDOFFILE");

stack.push("<Goal>");

int lineNumber;

while((!stack.empty()) && (!error))

{

Token = Lexer.GetNextToken();

lineNumber = Lexer.lineNumber-1;

error = Lexer.error;

currentToken = true;

while(currentToken && (!error)){

currentStack = stack.pop();

debugStep++;

holder = Token[0] + currentStack;

operation = parse.get(holder);

if (operation != null){

grammarOperations();

}

else if(Token[0].equals(UpperCase(currentStack))){

if (debugging)

{

DumpStack();

System.out.println("-> \*MATCH\* {consumes token}");

System.out.println();

}

currentToken = false;

}

else{

error = true;

System.out.println("Error: Expected " + currentStack + " before " + Token[0] +" in line " + lineNumber);

}

}

}

if((debugging) && !error)

System.out.println("! Accept !");

}

}