**PROBLEMS SPECIFICATION**DESIGN A LEXICAL ANALYZER GENERATOR (Lex)

**PROBLEM ANALYSIS**A Source Code is given as an input to the program. the program then breaks the source code into tokens that will later be accepted by a parser. The Java, C, C-Sharp and Basic languages were applied to aid this problem. the language consists of identifiers, keywords, operators and delimiters. the function of the program to break down the source code into identifiers, keywords, operators and delimiters and Literals. as the user inputs characters, the lex first inputs the character as a token until its able to recognize the group of characters under a particular specification in a specified language applied.

**INPUT**A HIGH LEVEL LANGUAGE SOURCE CODE:

**OUTPUT**  
THE VARIOUS LEXEMES OF THE INPUTED CODE (Tokens)

**ALGORITHM**STEP 1: Source Code File is Read by the Lex File Scanner.

: $sourceCode : File -> ReadFile(sourceArea :TextArea);

STEP 2: Construct all Tables into Lexs’ Table

: $TableOfKeys :Map<?,?> -> ConstructTable(table :File, pathtoFile :Path);

: $OPERATORS :List<?> -> ConstructOPTable(table :File, pathtoFile :Path);

: $DELIMETER :List<?> -> ConstructDelTable(table :File, pathtoFile :Path);

: $KeywordTable :List<?> -> ContructKeywords(table :File, pathtoFile :Path);

STEP 3: Convert each Line in the Source code to an Array of Characters containing each character in the line and create a List for acquiring Found words in source Code.

:$foundsList :List -> new ArrayList();

: for each $line :String in $sourceCode {$charline :Character[] -> ScanLine() ;}

STEP 4: Pick each Ordered Character in charline i.e Character array form of the line.

: for each $c :Character in $charline { $c -> getNextChar();}STEP 5: Is Selected character($c) a Whitespace ? YES -> Goto Step 4; NO -> Goto Step 6;

STEP 6: Is Selected character an Operator ? YES -> Goto Step 7; NO -> Goto Step 8;

:if OPERATORS.contains($c) – {return true(YES);} return false(NO);

STEP 7: Is index of character less than length of line and Next Char in line is an Operator ? YES ? -> Merge of this Character and next character an Operator, add to Found List ; NO -> Add this Character to Found List. 8: if character is a letter; save until next character is not a digit, letter or underscore then print saved character as :  
 i) keyword: if contained in the keyword array  
 2) letterstep 8: if character is in the delimiter array print the character as delimeterstep 9: if character is a number; save until 2nd character is not an operator then print saved character as operatorstep 10: if character is ‘ ” ’; save until next character is ‘ ” ’ then print saved character as stringstep 11: else print can’t find symbol

C

CODE :

package lexar;

import com.jfoenix.controls.JFXDecorator;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.effect.DropShadow;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javafx.stage.StageStyle;

/\*\*

\*

\* @author habuto

\*/

public class Lexar extends Application {

public static Stage stagee;

public static JFXDecorator windowPane;

@Override

public void start(Stage stage) throws Exception {

LexTableLoad load = new LexTableLoad();

stagee = stage;

stagee.initStyle(StageStyle.TRANSPARENT);

stagee.setTitle("True Lexar");

Parent root = FXMLLoader.load(getClass().getResource("Lexar.fxml"));

windowPane = new JFXDecorator(stage, root, false, false, true);

windowPane.setEffect(new DropShadow());

windowPane.setCustomMaximize(false);

Scene scene = new Scene(windowPane);

scene.setFill(Color.TRANSPARENT);

scene.getStylesheets().add(getClass().getResource("res/Lextyle.css").toExternalForm());

stage.setScene(scene);

stage.setResizable(false);

stage.show();

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

launch(args);

}

}

package lexar;

import com.jfoenix.controls.JFXComboBox;

import com.jfoenix.controls.JFXTextArea;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javafx.stage.FileChooser;

/\*\*

\*

\* @author habuto

\*/

public class LexDataControl implements ITables {

private File sourceFile;

private File tokenFile;

private static JFXTextArea sourceArea;

private static JFXTextArea tokenArea;

private static JFXComboBox<?> box;

public LexDataControl() {

}

public LexDataControl(JFXTextArea \_sourceArea, JFXTextArea \_tokenArea) {

sourceArea = \_sourceArea;

tokenArea = \_tokenArea;

box = LexarController.extendsBox;

}

private File loadSourceFile() {

FileChooser window = new FileChooser();

window.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("C | CSharp | Java | Text Files only ", EXTENSIONS));

File file = window.showOpenDialog(Lexar.stagee);

if (file != null) {

return file;

}

return null;

}

public File getSourceFile() {

return sourceFile;

}

public void setSourceFile() {

sourceFile = loadSourceFile();

}

public File getTokenFile() {

return tokenFile;

}

public void setTokenFile() {

tokenFile = loadTokenFile();

}

public void loadSourceFile(File dafile) {

if (LexarController.loadedfile) {

BufferedReader reader = null;

try {

reader = new BufferedReader(new FileReader(dafile));

String lineText;

sourceArea.setText("");

int LineNo = 0;

while ((lineText = reader.readLine()) != null) {

LineNo++;

sourceArea.appendText(lineText + "\n");

}

} catch (FileNotFoundException ex) {

Logger.getLogger(LexDataControl.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(LexDataControl.class.getName()).log(Level.SEVERE, null, ex);

} finally {

try {

if (reader != null) {

reader.close();

LexarController.loadedfile = false;

}

} catch (IOException ex) {

Logger.getLogger(LexDataControl.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

}

public void readSourceCode() {

File file = new File("src/temp.lex");

try {

FileWriter fwrite;

fwrite = new FileWriter(file);

fwrite.write(sourceArea.getText());

fwrite.close();

readSourceCode(file);

} catch (IOException ex) {

Logger.getLogger(LexDataControl.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void readSourceCode(File fl) throws FileNotFoundException, IOException {

try (BufferedReader reader = new BufferedReader(new FileReader(fl))) {

String lineText;

SOURCECODE.clear();

while ((lineText = reader.readLine()) != null) {

SOURCECODE.add(lineText);

}

}

}

private File loadTokenFile() {

FileChooser fileChooser = new FileChooser();

fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("Text File (\*.txt)","\*.txt"));

File token\_file = fileChooser.showSaveDialog(Lexar.stagee);

if (token\_file != null) {

return token\_file;

}

return null;

}

public void saveTokenFile(File dafile) {

try {

FileWriter filewriter;

filewriter = new FileWriter(dafile);

filewriter.write(tokenArea.getText());

filewriter.close();

} catch (IOException ex) {

Logger.getLogger(LexDataControl.class.getName()).log(Level.SEVERE, null, ex);

}

}

private String getFileExtension() {

String format = box.getAccessibleText();

String bend = "\*." + format.toLowerCase();

return bend;

}

private String getFileType() {

String format = box.getAccessibleText();

return format;

}

}

package lexar;

import java.io.\*;

import java.util.ArrayList;

import java.util.Map;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author habuto

\*/

public class LexTableLoad implements ITables, ILoader {

public LexTableLoad() {

initTable(PATHTOCKEYS, CKEYWORDS);

initTable(PATHTOCSKEYS, CSKEYWORDS);

initTable(PATHTOJAVAKEYS, JAVAKEYWORDS);

initTable(PATHTODELIMITERS, DELIMITERS);

initTable(PATHTOOPERATORS, OPERATORS);

initMap(PATHTOTABLE, KEYSMAP);

}

@Override

public void initTable(String path, ArrayList<String> table) {

try {

FileReader reader = new FileReader(path);

BufferedReader bread = new BufferedReader(reader);

String line;

while ((line = bread.readLine()) != null) {

table.add(line);

}

} catch (FileNotFoundException ex) {

Logger.getLogger(LexTableLoad.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(LexTableLoad.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

public void initMap(String path, Map<String, String> map) {

try {

FileReader reader = new FileReader(path);

BufferedReader breadMap = new BufferedReader(reader);

String line;

while ((line = breadMap.readLine()) != null) {

int mark = line.lastIndexOf(",");

map.put(getKey(mark, line), getValue(mark, line));

}

} catch (FileNotFoundException ex) {

Logger.getLogger(LexTableLoad.class.getName()).log(Level.SEVERE, null, ex);

} catch (IOException ex) {

Logger.getLogger(LexTableLoad.class.getName()).log(Level.SEVERE, null, ex);

}

}

private String getKey(int mark, String line) {

return line.substring(0, mark);

}

private String getValue(int mark, String line) {

return line.substring(mark + 1, line.length());

}

}

package lexar;

import com.jfoenix.controls.JFXTextArea;

import java.util.ArrayList;

/\*\*

\*

\* @author habuto

\*/

public class LexLogic implements ITables {

private static JFXTextArea sourceArea;

private static JFXTextArea tokenArea;

int t;

private static ArrayList<String> tokenizers;

public LexLogic(JFXTextArea \_source, JFXTextArea \_token) {

sourceArea = \_source;

tokenArea = \_token;

tokenizers = new ArrayList();

}

public LexLogic() {

}

public void runAnalysis() {

LexDataControl control = new LexDataControl(sourceArea, tokenArea);

control.readSourceCode();

tokenizers = new ArrayList();

for (int s = 0; s < SOURCECODE.size(); s++) {

char[] line = SOURCECODE.get(s).toCharArray();

String nnhold = "";

String literal = "";

String idhold = "";

ArrayList<String> foundList = new ArrayList();

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

String val = Character.toString(line[i]);

if (!Character.isWhitespace(line[i])) {

if (isOperator(val)) {

t = i;

foundList.add(scanOperator(val, line, t));

i = t;

} else if (Character.isDigit(line[i])) {

int c = i;

while (c < line.length && Character.isDigit(line[c])) {

nnhold += line[c];

i = c;

c++;

}

foundList.add(nnhold);

nnhold = "";

} else if (Character.isLetter(line[i]) || val.equals("\_")) {

int c = i;

while ((c < line.length) && (Character.isLetterOrDigit(line[c]) || Character.toString(line[c]).equals("\_"))) {

idhold += line[c];

i = c;

c++;

}

foundList.add(idhold);

idhold = "";

} else if (isDelimiter(val)) {

if (!isLiteral(val)) {

foundList.add(val);

} else if (ischarLit(val)) {

int k = i;

literal += val;

while ((i + 2) < line.length && k < (i + 2) && line[i + 2] == '\'') {

literal += line[k];

k++;

}

i = k;

foundList.add(literal);

literal = "";

} else if (isStrLit(val)) {

int c = i;

literal += val;

while ((c < line.length - 1) && !("\"".equals(Character.toString(line[c])))) {

literal += line[c];

i = c;

c++;

}

literal += ("\"".equals(Character.toString(line[c]))) ? line[c] : "";

foundList.add(literal);

literal = "";

}

}

}

}

tokenizers.add(setLineTokens(foundList, s + 1));

}

DisplayTokens(tokenizers);

}

private boolean isOperator(String ch) {

return OPERATORS.contains(ch);

}

private boolean isDelimiter(String ch) {

return DELIMITERS.contains(ch);

}

private boolean isLiteral(String ch) {

return (ch.equals("\"") || ch.equals("'"));

}

private boolean ischarLit(String ch) {

return ch.equals("'");

}

private boolean isStrLit(String ch) {

return ch.equals("\"");

}

private String scanOperator(String val, char[] line, int i) {

String ophold = val;

if ((i < line.length - 1) && isMergeableOP(i, line)) {

ophold += getNextChar(i, line);

++t;

}

return ophold;

}

private boolean isMergeableOP(int i, char[] \_line) {

String op1 = Character.toString(\_line[i]);

String op2 = Character.toString(\_line[i + 1]);

return OPERATORS.contains(op1 + op2);

}

private String setLineTokens(ArrayList<String> tokens, int lineNo) {

StringBuilder strbuild = new StringBuilder();

for (int i = 0; i < tokens.size(); i++) {

String tokensDisp;

if (KEYSMAP.containsKey(tokens.get(i))) {

tokensDisp = "[<" + lineNo + ">" + " : \"" + tokens.get(i) + "\" : " + KEYSMAP.get(tokens.get(i)) + "]\n";

} else if (isKeyWord(tokens.get(i))) {

tokensDisp = "[<" + lineNo + ">" + " : \"" + tokens.get(i) + "\" : {KEYWORD}]\n";

} else if (isNumber(tokens.get(i))) {

tokensDisp = "[<" + lineNo + ">" + " : \"" + tokens.get(i) + "\" : {NUMBER}]\n";

} else if (isTrueLiteral(tokens.get(i))) {

tokensDisp = "[<" + lineNo + ">" + " : \"" + tokens.get(i) + "\" : {LITERAL}]\n";

} else {

tokensDisp = "[<" + lineNo + ">" + " : \"" + tokens.get(i) + "\" : {ID}]\n";

}

strbuild.append("-");

strbuild.append(tokensDisp);

}

return strbuild.toString();

}

private char getNextChar(int i, char[] \_line) {

return \_line[i + 1];

}

private boolean isTrueLiteral(String str) {

return (str.charAt(0) == '"' || str.charAt(0) == '\'');

}

private boolean isKeyWord(String ch) {

return (JAVAKEYWORDS.contains(ch) || CKEYWORDS.contains(ch) || CSKEYWORDS.contains(ch));

}

private boolean isNumber(String s) {

boolean isnumber;

try {

Integer.parseInt(s);

isnumber = true;

} catch (NumberFormatException e) {

isnumber = false;

}

return isnumber;

}

private void DisplayTokens(ArrayList<String> tokensList) {

for (int i = 0; i < tokensList.size(); i++) {

tokenArea.appendText(tokensList.get(i) + "\n");

}

}

public ArrayList<String> getTokenizers() {

return tokenizers;

}

}

package lexar;

import com.jfoenix.controls.JFXButton;

import com.jfoenix.controls.JFXComboBox;

import com.jfoenix.controls.JFXTextArea;

import com.jfoenix.controls.JFXToggleButton;

import java.net.URL;

import java.util.ResourceBundle;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.input.KeyEvent;

/\*\*

\*

\* @author habuto

\*/

public class LexarController implements Initializable {

@FXML

private JFXTextArea sourceArea;

@FXML

private JFXTextArea tokenArea;

public static JFXComboBox<?> extendsBox;

@FXML

private JFXButton loadSourceBtn;

@FXML

private JFXButton analyseBtn;

@FXML

private JFXButton saveTokensBtn;

public static boolean loadedfile = false;

@FXML

private JFXButton clearTokenBtn;

@FXML

private JFXToggleButton liveAnalBtn;

@Override

public void initialize(URL url, ResourceBundle rb) {

liveAnalBtn.setSelected(false);

loadSourceBtn.setOnAction((ActionEvent e) -> {

loadedfile = true;

LexDataControl control = new LexDataControl(sourceArea, tokenArea);

control.setSourceFile();

if (control.getSourceFile() != null) {

control.loadSourceFile(control.getSourceFile());

}

});

analyseBtn.setOnAction((ActionEvent e) -> {

tokenArea.clear();

LexLogic logic = new LexLogic(sourceArea, tokenArea);

logic.runAnalysis();

});

clearTokenBtn.setOnAction((ActionEvent e) -> {

tokenArea.setText("");

});

saveTokensBtn.setOnAction((ActionEvent e) -> {

LexDataControl control = new LexDataControl(sourceArea, tokenArea);

control.setTokenFile();

if (control.getTokenFile() != null) {

control.saveTokenFile(control.getTokenFile());

}

});

sourceArea.setOnKeyReleased((KeyEvent event) -> {

if (liveAnalBtn.isSelected()) {

tokenArea.clear();

LexLogic logic = new LexLogic(sourceArea, tokenArea);

logic.runAnalysis();

} else {

}

});

}

@FXML

private void initialize(ActionEvent event) {

}

}

package lexar;

import java.util.ArrayList;

import java.util.Map;

/\*\*

\*

\* @author habuto

\*/

public interface ILoader {

public static final String PATHTOJAVAKEYS = "src/lexar/res/keywords\_java.txt";

public static final String PATHTOCKEYS = "src/lexar/res/keywords\_c.txt";

public static final String PATHTOCSKEYS = "src/lexar/res/keywords\_cs.txt";

public static final String PATHTOTABLE = "src/lexar/res/Table of Operators.txt";

public static final String PATHTOOPERATORS = "src/lexar/res/operators.txt";

public static final String PATHTODELIMITERS = "src/lexar/res/delimiters.txt";

public void initTable(String path, ArrayList<String> table);

public void initMap(String path, Map<String, String> map);

}

package lexar;

import java.util.\*;

/\*\*

\*

\* @author habuto

\*/

public interface ITables {

public static ArrayList<String> JAVAKEYWORDS = new ArrayList<>();

public static ArrayList<String> CKEYWORDS = new ArrayList<>();

public static ArrayList<String> CSKEYWORDS = new ArrayList<>();

public static ArrayList<String> OPERATORS = new ArrayList<>();

public static ArrayList<String> DELIMITERS = new ArrayList<>();

public static ArrayList<String> SOURCECODE = new ArrayList<>();

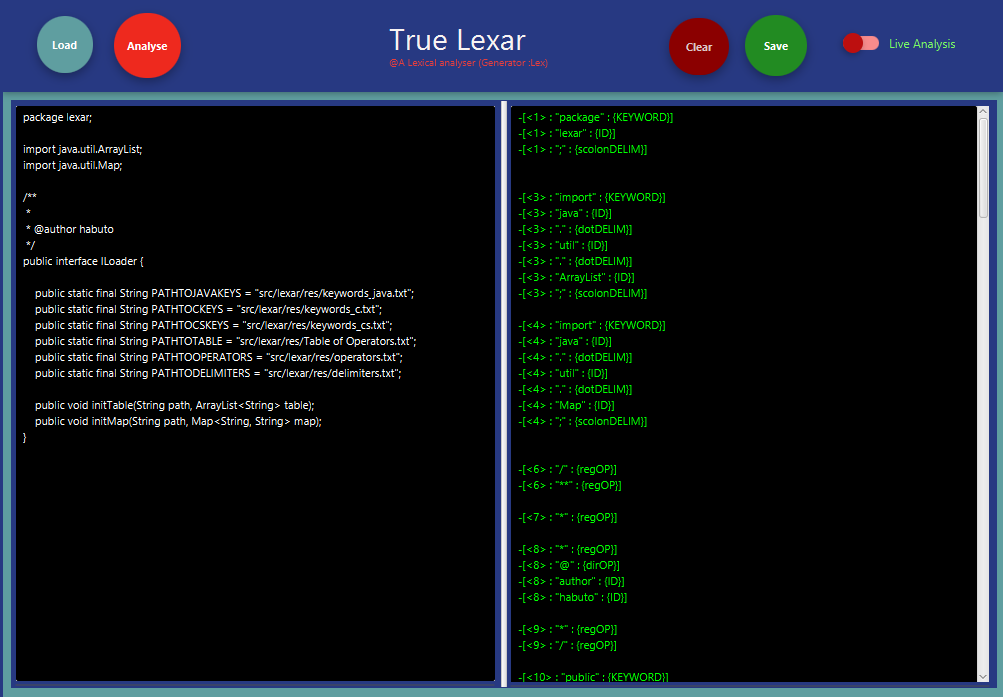
static String[] EXTSTRINGS = {"\*.c","\*.txt","\*.java","\*.cs","\*.TXT"};

public static Map<String,String> KEYSMAP = new HashMap<>();

public static List<String> EXTENSIONS = new ArrayList<>(Arrays.asList(EXTSTRINGS));

}

OUTPUT:



GRAMMAR TO CONVERT ROMAN NUMERALS TO ARABIC

S= > 0 | I| V|X|L|C| D| M | SS

I => +1 | -IV | -IX

V