***EXPERIMENT 4***

***AIM***: To remove Left Recursion from a grammar

***REQUIREMENTS:*** Visual Studio Code, Java Compiler.

***CODE***:

import java.util.ArrayList;

import java.util.Scanner;

public class LeftRecursion {

public static void removeLeftRecursion(ArrayList<Character> production) {

        if (production.get(0) != production.get(3)) {

            System.out.println("Left Recursion not present.");

            return;

        }

        char nonTerminal = production.get(0);

        int pipeLoc = 0;

        ArrayList<Character> alpha = new ArrayList<>();

        ArrayList<Character> beta = new ArrayList<>();

        for (int i = 4; i < production.size(); i++) {

            if (production.get(i) == '|') {

                pipeLoc = i;

                break;

            } else {

                alpha.add(production.get(i));

            }

        }

        for (int i = pipeLoc + 1; i < production.size(); i++) {

            beta.add(production.get(i));

        }

        StringBuilder a = new StringBuilder();

        for (Character character : alpha) {

            a.append(character);

        }

        StringBuilder b = new StringBuilder();

        for (Character character : beta) {

            b.append(character);

        }

        System.out.println("Production after removal of Left Recursion: ");

        System.out.println(nonTerminal + " -> " + b + nonTerminal + "'");

        System.out.println(nonTerminal + "' -> " + a + nonTerminal + "'|$");

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the production: ");

        String exp = scanner.nextLine();

        ArrayList<Character> production = new ArrayList<>();

        for (char c : exp.toCharArray()) {

            production.add(c);

        }

        removeLeftRecursion(production);

        scanner.close();

    }

}

***OUTPUT:***

A screen shot of a computer

Description automatically generated

***ERRORS:*** A screen shot of a computer screen

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 (To be scaled down to 1) | | |

***EXPERIMENT 5***

***AIM***: To remove left factoring from grammar.

***REQUIREMENTS:*** Visual Studio Code, Java Compiler.

***CODE***:

import java.util.ArrayList;

import java.util.Scanner;

public class LeftFactoring {

public static void removeLeftFactoring(ArrayList<Character> production) {

        char nonTerminal = production.get(0);

        int pipeLoc = 0;

        ArrayList<Character> firstHalf = new ArrayList<>();

        ArrayList<Character> secondHalf = new ArrayList<>();

        ArrayList<Character> alpha = new ArrayList<>();

        ArrayList<Character> beta = new ArrayList<>();

        ArrayList<Character> gamma = new ArrayList<>();

        for (int i = 3; i < production.size(); i++) {

            if (production.get(i) == '|') {

                pipeLoc = i;

                break;

            } else {

                firstHalf.add(production.get(i));

            }

        }

        for (int i = pipeLoc + 1; i < production.size(); i++) {

            secondHalf.add(production.get(i));

        }

        int x = 0;

        while (x < firstHalf.size() && x < secondHalf.size() && firstHalf.get(x) == secondHalf.get(x)) {

            alpha.add(firstHalf.get(x));

            x++;

        }

        if (alpha.size() == 0) {

            System.out.println("Left Factoring not present.");

            return;

        }

        for (int i = x; i < firstHalf.size(); i++) {

            beta.add(firstHalf.get(i));

        }

        for (int i = x; i < secondHalf.size(); i++) {

            gamma.add(secondHalf.get(i));

        }

        StringBuilder a = new StringBuilder();

        for (Character character : alpha) {

            a.append(character);

        }

        String b = beta.isEmpty() ? "$" : beta.toString();

        String y = gamma.isEmpty() ? "$" : gamma.toString();

        System.out.println("Production after removal of Left Factoring: ");

        System.out.println(nonTerminal + "-> " + a + nonTerminal + "'");

        System.out.println(nonTerminal + "'-> " + b + "|" + y);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the production: ");

        String exp = scanner.nextLine();

        ArrayList<Character> production = new ArrayList<>();

        for (char c : exp.toCharArray()) {

            production.add(c);

        }

        removeLeftFactoring(production);

        scanner.close();

    }

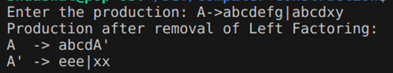
}

***OUTPUT:***

A screen shot of a computer

Description automatically generated

***ERRORS:***



A computer screen shot of a black background

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 (To be scaled down to 1) | | |

***EXPERIMENT 6***

***AIM***: To calculate First and Follow set from a given CFG.

***REQUIREMENTS:*** Visual Studio Code, Java Compiler.

***CODE***:

import java.io.File;

import java.io.FileNotFoundException;

import java.util.\*;

public class FirstFollow {

    static Set<Character> ss = new HashSet<>();

    static boolean dfs(char i, char org, char last, Map<Character, List<List<Character>>> mp) {

        boolean rtake = false;

        for (List<Character> r : mp.get(i)) {

            boolean take = true;

            for (Character s : r) {

                if (s == i) break;

                if (!take) break;

                if (!(s >= 'A' && s <= 'Z') && s != 'e') {

                    ss.add(s);

                    break;

                } else if (s == 'e') {

                    if (org == i || i == last)

                        ss.add(s);

                    rtake = true;

                    break;

                } else {

                    take = dfs(s, org, r.get(r.size() - 1), mp);

                    rtake |= take;

                }

            }

        }

        return rtake;

    }

    public static void main(String[] args) {

        Scanner scanner;

        try {

            scanner = new Scanner(new File("06-input-file.txt"));

        } catch (FileNotFoundException e) {

            System.out.println("File not found!");

            return;

        }

        String num;

        Map<Character, List<List<Character>>> mp = new HashMap<>();

        boolean flag = false;

        char start = ' ';

        System.out.println("Grammar: ");

        while (scanner.hasNextLine()) {

            num = scanner.nextLine();

            if (!flag) {

                start = num.charAt(0);

                flag = true;

            }

            System.out.println(num);

            char s = num.charAt(0);

            List<List<Character>> temp = new ArrayList<>();

            List<Character> innerTemp = new ArrayList<>();

            for (int i = 3; i < num.length(); i++) {

                if (num.charAt(i) == '|') {

                    temp.add(innerTemp);

                    innerTemp = new ArrayList<>();

                } else {

                    innerTemp.add(num.charAt(i));

                }

            }

            temp.add(innerTemp);

            mp.put(s, temp);

        }

        Map<Character, Set<Character>> fmp = new HashMap<>();

        for (Map.Entry<Character, List<List<Character>>> q : mp.entrySet()) {

            ss.clear();

            dfs(q.getKey(), q.getKey(), q.getKey(), mp);

            Set<Character> set = new HashSet<>(ss);

            fmp.put(q.getKey(), set);

        }

        System.out.println();

        System.out.println("FIRST: ");

        for (Map.Entry<Character, Set<Character>> q : fmp.entrySet()) {

            StringBuilder ans = new StringBuilder();

            ans.append(q.getKey()).append(" = {");

            for (char r : q.getValue()) {

                ans.append(r).append(',');

            }

            if (!q.getValue().isEmpty()) {

                ans.deleteCharAt(ans.length() - 1);

            }

            ans.append("}");

            System.out.println(ans);

        }

        Map<Character, Set<Character>> gmp = new HashMap<>();

        gmp.put(start, new HashSet<>(Collections.singletonList('$')));

        int count = 10;

        while (count-- > 0) {

            for (Map.Entry<Character, List<List<Character>>> q : mp.entrySet()) {

                for (List<Character> r : q.getValue()) {

                    for (int i = 0; i < r.size() - 1; i++) {

                        char currentChar = r.get(i);

                        if (currentChar >= 'A' && currentChar <= 'Z') {

                            char nextChar = r.get(i + 1);

                            if (!(nextChar >= 'A' && nextChar <= 'Z')) {

                                gmp.computeIfAbsent(currentChar, k -> new HashSet<>()).add(nextChar);

                            } else {

                                char temp = nextChar;

                                int j = i + 1;

                                while (temp >= 'A' && temp <= 'Z') {

                                    if (fmp.get(temp).contains('e')) {

                                        for (char g : fmp.get(temp)) {

                                            if (g != 'e') {

                                                gmp.computeIfAbsent(currentChar, k -> new HashSet<>()).add(g);

                                            }

                                        }

                                        j++;

                                        if (j < r.size()) {

                                            temp = r.get(j);

                                            if (!(temp >= 'A' && temp <= 'Z')) {

                                                gmp.computeIfAbsent(currentChar, k -> new HashSet<>()).add(temp);

                                                break;

                                            }

                                        } else {

                                            for (char g : gmp.get(q.getKey())) {

                                                gmp.computeIfAbsent(currentChar, k -> new HashSet<>()).add(g);

                                            }

                                            break;

                                        }

                                    } else {

                                        for (char g : fmp.get(temp)) {

                                            gmp.computeIfAbsent(currentChar, k -> new HashSet<>()).add(g);

                                        }

                                        break;

                                    }

                                }

                            }

                        }

                    }

                    char lastChar = r.get(r.size() - 1);

                    if (lastChar >= 'A' && lastChar <= 'Z') {

                        for (char g : gmp.get(q.getKey())) {

                            gmp.computeIfAbsent(lastChar, k -> new HashSet<>()).add(g);

                        }

                    }

                }

            }

        }

        System.out.println();

        System.out.println("FOLLOW: ");

        for (Map.Entry<Character, Set<Character>> q : gmp.entrySet()) {

            StringBuilder ans = new StringBuilder();

            ans.append(q.getKey()).append(" = {");

            for (char r : q.getValue()) {

                ans.append(r).append(',');

            }

            if (!q.getValue().isEmpty()) {

                ans.deleteCharAt(ans.length() - 1);

            }

            ans.append("}");

            System.out.println(ans);

        }

        scanner.close();

    }

}

***INPUT FILE:***

A screenshot of a computer program

Description automatically generated

***OUTPUT:***

A computer screen shot of a black screen

Description automatically generated

***ERRORS:***

A screen shot of a computer

Description automatically generated A screenshot of a computer screen

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 (To be scaled down to 1) | | |

***OPEN ENDED EXPERIMENT***

***AIM***: To implement a basic compiler.

***Language Description:***

The language has 5 keywords: DECLARE, SET, PRINT, DEL, QUIT.

The language has 2 operators: = and +

The variable declaration in this language is limited to small letters of the English alphabet, such as: a, b, c, d, etc.

Whitespace acts as a delimiter, and a newline is considered the end of line. Only a single instruction is allowed per line.

This language is capable of arithmetic addition of positive integer values.

**=>DECLARE**

The DECLARE keyword initializes a variable with the value of 0.

Syntax: DECLARE a

**=>SET**

The SET keyword updates the value of a variable. The value must be a positive integer. The SET keyword can also be used to assign values to a variable that is equal to the sum of the two other variables. Note, arithmetic operations using the SET keyword only allows addition between variables.

If the SET keyword is used with an undeclared variable, an error is thrown, and program execution stops.

Syntax: SET a = 5

Syntax: SET a = b + c

**=>PRINT**

The PRINT keyword is followed by a variable. The value of the variable is written to the output file.

If the PRINT keyword is used with an undeclared variable, an error is thrown, and program execution stops.

Syntax: PRINT a

**=>DEL**

The DEL keyword deletes the variable.

If the DEL keyword is used with an undeclared variable, an error is thrown and program execution stops.

Syntax: DEL a

**=>QUIT**

As soon as the QUIT keyword is encountered by itself in a newline, program execution stops.

Syntax: QUIT

If a syntax error occurs anywhere, an error is thrown, and program execution stops.

***PSEUDOCODE***:

The language keywords and operators are specified.

2. The language syntax is established.

3. Reading from input file and writing to output file is enabled.

4. Two hashmaps are created, one keeps track of variables and their value. And the other keeps track of the occurrence of each variable. Together, they act as the symbol table.

5. At the initial file read, tokenization and categorization are carried out.

6. The final pointer is reset, and the input file is read a second time. This time syntax and semantic analysis takes place, and if everything is in order, execution takes place. Else, an error is thrown, and program execution stops.

7. The output is written in the output file.

8. Reading from input file and writing to output file is disabled.

9. The number of tokens and their category is printed out in the console.

10. EXIT

***CODE:***

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Scanner;

import java.util.Set;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class LanguageConverter {

    private static final Set<String> langKeywords = new HashSet<>();

    private static final Set<String> langOperators = new HashSet<>();

    private static final Map<Character, Integer> variable = new HashMap<>();

    static {

        langKeywords.add("DECLARE");

        langKeywords.add("SET");

        langKeywords.add("PRINT");

        langKeywords.add("DEL");

        langKeywords.add("QUIT");

        langOperators.add("+");

        langOperators.add("=");

    }

    private static final Pattern declarePattern = Pattern.compile("^DECLARE [a-z]");

    private static final Pattern setEqualToNumPattern = Pattern.compile("^SET [a-z] [=] \\d+(\\.\\d+)?");

    private static final Pattern setEqualToSumPattern = Pattern.compile("^SET [a-z] [=] [a-z] [+] [a-z]");

    private static final Pattern printPattern = Pattern.compile("^PRINT [a-z]");

    private static final Pattern delPattern = Pattern.compile("^DEL [a-z]");

    public static void main(String[] args) {

        File inputFile = new File("open-ended-input-02.txt");

        File outputFile = new File("open-ended-output.txt");

        try (Scanner scanner = new Scanner(inputFile);

                FileWriter writer = new FileWriter(outputFile)) {

            int count = 0;

            int keywords = 0;

            int operators = 0;

            int varCount = 0;

            Map<Character, Integer> variableInstances = new HashMap<>();

            while (scanner.hasNextLine()) {

                String line = scanner.nextLine().trim();

                String[] tokens = line.split("\\s+");

                for (String token : tokens) {

                    if (langKeywords.contains(token)) {

                        keywords++;

                    } else if (langOperators.contains(token)) {

                        operators++;

                    } else if (token.length() == 1 && Character.isLowerCase(token.charAt(0))) {

                        char ch = token.charAt(0);

                        if (!variableInstances.containsKey(ch)) {

                            variableInstances.put(ch, 1);

                            varCount++;

                        }

                    }

                    count++;

                }

            }

            // scanner.close();

            // scanner.reset();

            variable.clear();

            while (scanner.hasNextLine()) {

                String line = scanner.nextLine().trim();

                Matcher declareMatcher = declarePattern.matcher(line);

                Matcher setEqualToNumMatcher = setEqualToNumPattern.matcher(line);

                Matcher setEqualToSumMatcher = setEqualToSumPattern.matcher(line);

                Matcher printMatcher = printPattern.matcher(line);

                Matcher delMatcher = delPattern.matcher(line);

                if (declareMatcher.matches()) {

                    char temp = line.charAt(8);

                    variable.put(temp, 0);

                } else if (setEqualToNumMatcher.matches()) {

                    char temp = line.charAt(4);

                    if (variable.containsKey(temp)) {

                        int val = Integer.parseInt(line.substring(8));

                        variable.put(temp, val);

                    } else {

                        writer.write("ERROR: Attempted to assign value to undeclared variable.\n");

                        break;

                    }

                } else if (setEqualToSumMatcher.matches()) {

                    char temp = line.charAt(4);

                    char var1 = line.charAt(8);

                    char var2 = line.charAt(12);

                    if (variable.containsKey(temp) && variable.containsKey(var1) && variable.containsKey(var2)) {

                        variable.put(temp, variable.get(var1) + variable.get(var2));

                    } else {

                        writer.write("ERROR: Undeclared variable involved in addition.\n");

                        break;

                    }

                } else if (printMatcher.matches()) {

                    char temp = line.charAt(6);

                    if (variable.containsKey(temp)) {

                        writer.write(variable.get(temp) + "\n");

                    } else {

                        writer.write("ERROR: Attempted to print undeclared variable.\n");

                        break;

                    }

                } else if (delMatcher.matches()) {

                    char temp = line.charAt(4);

                    if (variable.containsKey(temp)) {

                        variable.remove(temp);

                    } else {

                        writer.write("ERROR: Attempted to delete non-existent variable.\n");

                        break;

                    }

                } else {

                    writer.write("ERROR: Illegible Instruction\n");

                    break;

                }

                if (line.equals("QUIT")) {

                    break;

                }

            }

            System.out.println("Total number of tokens: " + count);

            System.out.println("Total number of keywords: " + keywords);

            System.out.println("Total number of operators: " + operators);

            System.out.println("Total number of unique variables: " + varCount);

        } catch (FileNotFoundException e) {

            System.out.println("File not found!");

        } catch (IOException e) {

            System.out.println("Error writing to file!");

        }

    }

}

***INPUT FILE 1 INPUT FILE 2***

***A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated***

***OUTPUT:***

**INPUT FILE 1**

A screen shot of a computer

Description automatically generated

**INPUT FILE 2**

A screen shot of a computer code

Description automatically generated