**Source Code**

package first\_follow;

import com.google.common.collect.Table;

import com.google.common.collect.HashBasedTable;

import java.io.\*;

import java.util.\*;

import java.util.Map.Entry;

import java.util.stream.Collectors;

import java.util.stream.Stream;

import org.apache.commons.lang3.StringUtils;

class TableBuilder

{

List<String[]> rows = new LinkedList<String[]>();

public void addRow(String... cols)

{

rows.add(cols);

rows.add(new String[]{" "});

}

@SuppressWarnings("deprecation")

private int[] colWidths()

{

int cols = -1;

for(String[] row : rows)

cols = Math.max(cols, row.length);

int[] widths = new int[cols];

for(String[] row : rows) {

for(int colNum = 0; colNum < row.length; colNum++) {

widths[colNum] =

Math.max(

widths[colNum],

StringUtils.length(row[colNum]))+2;

}

}

return widths;

}

@Override

public String toString()

{

StringBuilder buf = new StringBuilder();

int[] colWidths = colWidths();

for(String[] row : rows) {

for(int colNum = 0; colNum < row.length; colNum++) {

buf.append(

StringUtils.rightPad(

StringUtils.defaultString(

row[colNum]), colWidths[colNum]));

buf.append(' ');

}

buf.append('\n');

}

return buf.toString();

}

}

public class Project1 {

static ArrayList <String> list = new ArrayList<String>();

static Map <String,ArrayList<String>> follow = new LinkedHashMap<String,ArrayList<String>>();

static ArrayList <String> rhs = new ArrayList<String>();

static Map <String,ArrayList<String>> lhs = new LinkedHashMap<String,ArrayList<String>>();

static Map <String,Integer> lhs\_rule\_no = new LinkedHashMap<String,Integer>();

static ListIterator<String> itr;

static Table<String, String, String> table = HashBasedTable.create();

static Map<String,String> code\_table = new HashMap<String,String>();

static Map<String,String> symbol\_table = new HashMap<String,String>();

public static String table\_creation (String string)

{

String string2 ;

code\_table.put("int", "keyword");

code\_table.put("float", "keyword");

code\_table.put("void", "keyword");

code\_table.put("if", "keyword");

code\_table.put("else", "keyword");

code\_table.put("for", "keyword");

code\_table.put("read", "keyword");

code\_table.put("print", "keyword");

code\_table.put("main", "keyword");

code\_table.put("{", "punctuation");

code\_table.put("}", "punctuation");

code\_table.put("(", "punctuation");

code\_table.put(")", "punctuation");

code\_table.put(";", "punctuation");

code\_table.put(",", "punctuation");

code\_table.put("=", "assign-op");

code\_table.put("+", "arith-op");

code\_table.put("-", "arith-op");

code\_table.put("/", "arith-op");

code\_table.put("\*", "arith-op");

code\_table.put("<", "rel-op");

code\_table.put(">", "rel-op");

code\_table.put("==", "rel-op");

code\_table.put("<=", "rel-op");

code\_table.put(">=", "rel-op");

code\_table.put("$", "delimeter");

ArrayList <String> newly = new ArrayList<String>();

for(String retval : string.split("\\s+"))

{

retval = retval.trim();

if(!code\_table.containsKey(retval))

{

if(isNumeric(retval))

{

newly.add("integer");

symbol\_table.put(retval, "Integer");

}

else

{

newly.add("id");

symbol\_table.put(retval, "Indentifier");

}

}

else

newly.add(retval);

}

string2 = String.join(" ", newly);

return string2;

}

public static boolean isNumeric(String s) {

return s.matches("[-+]?\\d\*\\.?\\d+");

}

public static ArrayList <String> findallsubstring(String st)

{

ArrayList <String> allsubstring = new ArrayList <String>();

Collections.addAll(allsubstring, st.split("\\s+"));

return allsubstring;

}

public static void follow(String str)

{

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

{

for(String retval: rhs.get(i).split("\\|"))

{

retval=retval.trim();

if(retval.contains(str))

{

if(retval.indexOf(str)+str.length()==retval.length())

{

if(list.get(i).split("->")[0].trim().compareTo(str)!=0)

{

if(follow.get(list.get(i).split("->")[0].trim()).isEmpty())

follow(list.get(i).split("->")[0].trim());

follow.get(str).addAll(follow.get(list.get(i).split("->")[0].trim()));

}

else

break;

}

else

{

int count = retval.substring(retval.indexOf(str)+str.length()).trim().split("\\s+").length;

int current=0;

for(String retval2 : retval.substring(retval.indexOf(str)+str.length()).trim().split("\\s+"))

{

current++;

retval2 = retval2.trim();

if(lhs.containsKey(retval2))

{

follow.get(str).addAll(lhs.get(retval2));

if(!lhs.get(retval2).contains("^"))

break;

else if(lhs.get(retval2).contains("^") && count==current)

{

if(list.get(i).split("->")[0].trim().compareTo(str)!=0)

{

if(follow.get(list.get(i).split("->")[0].trim()).isEmpty())

follow(list.get(i).split("->")[0].trim());

follow.get(str).addAll(follow.get(list.get(i).split("->")[0].trim()));

}

}

}

else

{

follow.get(str).add(retval2);

break;

}

}

}

}

}

}

if(str.compareTo(list.get(list.size()-1).split("->")[0].trim())==0)

follow.get(str).add("$");

}

public static void first(String str,int k)

{

if(!lhs.get(str).isEmpty())

return;

boolean flag;

for(String retval: rhs.get(k).split("\\|"))

{

flag=true;

retval=retval.trim();

Iterator<Map.Entry<String,ArrayList<String>>> it1;

ArrayList <String> substring = new ArrayList<String>();

substring = findallsubstring(retval.trim());

Iterator<String> it2 = substring.iterator();

int token\_count=0;

while(it2.hasNext())

{

token\_count++;

String str1 = it2.next();

flag=true;

it1 = lhs.entrySet().iterator();

while(it1.hasNext())

{

Entry<String, ArrayList<String>> map = it1.next();

if(str1.equals(map.getKey()))

{

if(token\_count==1)

{

if(map.getValue().isEmpty())

first(map.getKey(),lhs\_rule\_no.get(map.getKey()));

lhs.get(str).addAll(map.getValue());

flag=false;

break;

}

else if(token\_count>1)

{

if(map.getValue().isEmpty())

first(map.getKey(),lhs\_rule\_no.get(map.getKey()));

if (!(lhs.get(str).lastIndexOf("^")==lhs.get(str).size()-1))

{

flag=false;

break;

}

lhs.get(str).addAll(map.getValue());

flag=false;

break;

}

}

}

if(flag)

{

lhs.get(str).add(str1);

break;

}

else if (!(lhs.get(str).lastIndexOf("^")==lhs.get(str).size()-1))

break;

}

if(Collections.frequency(lhs.get(str), "^")==substring.size())

{

lhs.get(str).removeAll(Collections.singleton("^"));

lhs.get(str).add("^");

}

}

}

public static void parser(String st)

{

StringBuffer pop;

String temporary1 = st.trim();

Stack<StringBuffer> stack = new Stack<StringBuffer>();

stack.push(new StringBuffer("$"));

stack.push(new StringBuffer(list.get(list.size()-1).split("->")[0].trim()));

System.out.printf( " \n %10s %280s %118s","Stack","Input-Buffer","Action");

System.out.print("\n\n");

for(String retval : st.split("\\s+"))

{

retval = retval.trim();

do

{

temporary1 = temporary1.substring(temporary1.indexOf(retval));

System.out.printf(" %-180s %180s",stack.toString(),temporary1);

pop = stack.pop();

if(retval.equals(pop.toString()))

{

System.out.format("%" + 35 + "s" ,"matched");

System.out.println();

break;

}

if(table.get(pop.toString(),retval.toString())!=null)

{

StringBuffer sb = new StringBuffer(table.get(pop.toString(),retval.toString()));

System.out.printf("%28s -> %-4s",pop.toString(),sb.toString());

System.out.println();

if(!sb.equals("^"))

{

for(String temporary : sb.reverse().toString().split("\\s+"))

{

if(temporary.length()>1)

stack.push(new StringBuffer(temporary).reverse());

else

stack.push(new StringBuffer(temporary));

}

}

if(stack.peek().toString().equals("^"))

stack.pop();

}

else

{

System.out.println("\n \t Wrong , there is no rule like : " + pop.toString() + "->" + retval);

return;

}

}while(true);

}

System.out.println("Correct");

}

public static void create\_parse\_table(List<String> l1,List<String> l2)

{

Iterator<String> i1 = rhs.iterator();

while(i1.hasNext())

{

String temp = i1.next();

int index = rhs.indexOf(temp);

for(String retval: temp.split("\\|"))

{

retval = retval.trim();

if(l1.contains(retval.split(" ")[0]))

table.put(list.get(index).split("->")[0].trim(), retval.split(" ")[0], retval);

else if(l2.contains(retval.split(" ")[0]))

{

Iterator<String> i2 = lhs.get(retval.split(" ")[0]).iterator();

while(i2.hasNext())

{

String temp2 = i2.next();

if(temp2.equals("^"))

{

Iterator<String> i3 = follow.get(retval.split(" ")[0]).iterator();

while(i3.hasNext())

table.put(list.get(index).split("->")[0].trim(), i3.next(), retval);

}

else

table.put(list.get(index).split("->")[0].trim(), temp2, retval);

}

}

else

{

Iterator<String> i3 = follow.get(list.get(index).split("->")[0].trim()).iterator();

while(i3.hasNext())

{

String tempo = i3.next();

table.put(list.get(index).split("->")[0].trim(), tempo, retval);

}

}

}

}

}

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

FileInputStream in = new FileInputStream("/Users/anuragroy/eclipse-workspace/CompilerLab/src/first\_follow/cfg2.txt");

BufferedReader reader = new BufferedReader(new InputStreamReader(in));

PrintWriter out = new PrintWriter("/Users/anuragroy/eclipse-workspace/CompilerLab/src/first\_follow/output.txt");

String str;

int k=0;

Iterator<Map.Entry<String,ArrayList<String>>> it;

String s1 = reader.readLine();

while(s1 != null){

list.add(s1);

s1 = reader.readLine();

}

reader.close();

Collections.reverse(list);

itr = list.listIterator();

while(itr.hasNext())

{

str = itr.next();

lhs.put(str.split("->")[0].trim(),new ArrayList<String>());

follow.put(str.split("->")[0].trim(),new ArrayList<String>());

lhs\_rule\_no.put(str.split("->")[0].trim(),k);

rhs.add(str.split("->")[1].trim());

k++;

}

k=0;

it = lhs.entrySet().iterator();

while(it.hasNext())

{

first(it.next().getKey(),k);

k++;

}

it = lhs.entrySet().iterator();

while(it.hasNext())

{

Entry<String, ArrayList<String>> map = it.next();

if(lhs.get(map.getKey()).lastIndexOf("^")!=lhs.get(map.getKey()).size()-1)

lhs.get(map.getKey()).removeAll(Arrays.asList("^"));

ArrayList <String> temp = new ArrayList<String>();

temp = (ArrayList<String>) map.getValue().stream().distinct().collect(Collectors.toList());

lhs.get(map.getKey()).clear();

lhs.get(map.getKey()).addAll(temp);

out.println("First of (" + map.getKey() + ") = " + map.getValue());

}

out.println();

it = lhs.entrySet().iterator();

while(it.hasNext())

{

follow(it.next().getKey());

}

it = follow.entrySet().iterator();

while(it.hasNext())

{

Entry<String, ArrayList<String>> map = it.next();

if(follow.get(map.getKey()).lastIndexOf("^")!=follow.get(map.getKey()).size()-1)

follow.get(map.getKey()).removeAll(Arrays.asList("^"));

follow.get(map.getKey()).removeAll(Arrays.asList("'"));

ArrayList <String> temp = new ArrayList<String>();

temp = (ArrayList<String>) map.getValue().stream().distinct().collect(Collectors.toList());

follow.get(map.getKey()).clear();

follow.get(map.getKey()).addAll(temp);

out.println("Follow of (" + map.getKey() + ") = " + map.getValue());

}

List<String> list\_of\_terminals = rhs.stream().map(d -> d.split(" ")).flatMap(e -> Arrays.stream(e)).filter(e -> lhs.containsKey(e)==false && !e.equals("|") && !e.equals("^")).distinct().collect(Collectors.toList());

list\_of\_terminals.add("$");

List<String> list\_of\_non\_terminals = lhs.keySet().stream().collect(Collectors.toList());

create\_parse\_table(list\_of\_terminals,list\_of\_non\_terminals);

TableBuilder tb = new TableBuilder();

tb.addRow(Stream.concat(Stream.of(" "),list\_of\_terminals.stream()).toArray(String[]::new));

List<String> temporary;

for(int z = 0;z <list\_of\_non\_terminals.size(); z++)

{

temporary = new LinkedList<String>();

temporary.add(list\_of\_non\_terminals.get(z));

for(int y = 0;y <list\_of\_terminals.size(); y++)

{

if(table.get(list\_of\_non\_terminals.get(z), list\_of\_terminals.get(y))==null)

temporary.add("-");

else

temporary.add(list\_of\_non\_terminals.get(z) +" -> "+ table.get(list\_of\_non\_terminals.get(z), list\_of\_terminals.get(y)));

}

tb.addRow(temporary.stream().toArray(String[]::new));

}

System.out.println(tb.toString());

parser(table\_creation("void main ( ) { int a , b , k ; read ( k ) ; print ( a ) ; if ( b == 5 ) a = b \* k ; else a = b ; } $"));

out.close();

}

}