МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Лабораторна робота №8

з курсу

«Інженерія даних та знань»

*на тему:*

**«Аналіз складних висловлювань»**

Виконав:

ст. гр. ITIT-12

Кобак Р.З.

Прийняла:

Рибчак З.Л.

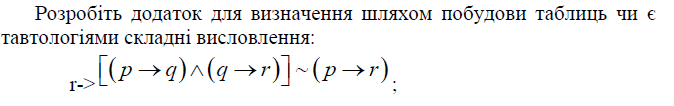
**Львів 2017**

**Мета роботи:**

Вивчення алгоритмів аналізу складних виразів логіки висловлювань та їх застосування для розв’язування задач визначення тавтологій.

**Індивідуальне завдання:**

**Варіант 3**



**Порядок виконання:**

Реалізовано додаток, для роботи з виразами Булевої логіки.  
Основні функції розробленого додатку:

1. Зчитування виразів Булевої логіки
2. Перетворення виразів в зворотний польський запис з урахуванням пріоритетів операцій.
3. Обчислення виразів
4. Побудова Таблиці істинності виразів
5. Перевірка чи даний вираз є тавтологією.

**Код програми:**

**public class** Expression {  
  
 **public** Expression(String s) {  
 **for** (String symbol : s.split(**" "**)) {  
 **if** (OperationType.*isOperator*(symbol)) {  
 **symbolList**.add(**new** Symbol(OperationType.*parse*(symbol)));  
 } **else** {  
 **symbolList**.add(**new** Symbol(**new** Atom(String.*valueOf*(symbol))));  
 }  
 }  
 }  
  
 **private** List<Symbol> **symbolList** = **new** ArrayList<>();  
 **private** List<Symbol> **postfixNotation** = **new** ArrayList<>();  
  
 **public void** addAtom(Atom atom) {  
 **symbolList**.add(**new** Symbol(atom));  
 }  
  
 **public void** addOperation(OperationType type) {  
 **symbolList**.add(**new** Symbol(type));  
 }  
  
 **public void** removeLast() {  
 **if** (!**symbolList**.isEmpty()) {  
 **symbolList**.remove(**symbolList**.size() - 1);  
 }  
 }  
  
 **public void** clear() {  
 **symbolList**.clear();  
 }  
  
 **public** List<Atom> getAtoms() {  
 List<Atom> atoms = **new** ArrayList<>();  
  
 **for** (Symbol symbol : **symbolList**) {  
 **if** (symbol.isAtom() && !atoms.contains(symbol.**atom**)) {  
 atoms.add(symbol.**atom**);  
 }  
 }  
  
 **return** atoms;  
 }  
   
 **public boolean** getResult()**throws** Exception {  
 **postfixNotation** = generatePostfixNotation();  
 Stack<Boolean> calculationStack = **new** Stack<>();  
  
 **for** (Symbol symbol : **postfixNotation**) {  
 **if** (symbol.isAtom()) {  
 calculationStack.push(symbol.**atom**.**value**);  
 } **else if** (!calculationStack.isEmpty()) {  
 **switch** (symbol.**operationType**) {  
 **case *Not***:  
 calculationStack.push(!calculationStack.pop());  
 **break**;  
  
 **case *And***:  
 calculationStack.push(calculationStack.pop() && calculationStack.pop());  
 **break**;  
  
 **case *Or***:  
 calculationStack.push(calculationStack.pop() || calculationStack.pop());  
 **break**;  
  
 **case *Equals***:  
 calculationStack.push(calculationStack.pop() == calculationStack.pop());  
 **break**;  
  
 **case *Imp***:  
 calculationStack.push(calculationStack.pop() || !calculationStack.pop());  
 **break**;  
 }  
 }  
 }  
  
 **return** calculationStack.isEmpty() ? **false** : calculationStack.pop();  
 }  
  
 **public boolean**[][] getTruthTable() **throws** Exception {  
 List<Atom> atoms = getAtoms();  
  
 **int** columnCount = atoms.size() + 1;  
 **int** rowsCount = (**int**) Math.*pow*(2, atoms.size());  
 **boolean**[][] table = **new boolean**[rowsCount][columnCount];  
  
  
 **int** divisor = 1;  
  
 **for** (**int** i = 0; i < columnCount; i++) {  
 divisor \*= 2;  
 **int** counter = 1;  
 **boolean** cellValue = **true**;  
  
 **for** (**int** j = 0; j < rowsCount; j++) {  
 **if** (counter > rowsCount / divisor) {  
 counter = 1;  
 cellValue = !cellValue;  
 }  
  
 table[j][i] = cellValue;  
 counter++;  
 }  
 }  
  
 **boolean**[] savedValues = **new boolean**[atoms.size()];  
  
 **for** (**int** i = 0; i < atoms.size(); i++) {  
 savedValues[i] = atoms.get(i).**value**;  
 }  
  
 **for** (**int** i = 0; i < rowsCount; i++) {  
 **for** (**int** j = 0; j < atoms.size(); j++) {  
 atoms.get(j).**value** = table[i][j];  
 }  
  
 table[i][columnCount - 1] = getResult();  
 }  
  
 **for** (**int** i = 0; i < atoms.size(); i++) {  
 atoms.get(i).**value** = savedValues[i];  
 }  
  
 **return** table;  
 }  
  
 **private** List<Symbol> generatePostfixNotation() **throws** Exception{  
 List<Symbol> postfixNotation = **new** ArrayList<>();  
  
 ArrayDeque<Symbol> operationStack = **new** ArrayDeque<>();  
 Symbol inputSymbol;  
 Symbol tempSymbol;  
  
 **for** (**int** i = 0; i < **symbolList**.size(); i++) {  
 inputSymbol = **symbolList**.get(i);  
 **if** (inputSymbol.isOperation()) {  
 **while** (!operationStack.isEmpty()) {  
 tempSymbol = operationStack.peekLast();  
 **if** (tempSymbol.isOperation() && (inputSymbol.**operationType**.getPriority() <= tempSymbol.**operationType**.getPriority())) {  
 postfixNotation.add(tempSymbol);  
 operationStack.removeLast();  
 } **else** {  
 **break**;  
 }  
 }  
 operationStack.offer(inputSymbol);  
 } **else if** (OperationType.***LBracket*** == inputSymbol.**operationType**) {  
 operationStack.offerLast(inputSymbol);  
 } **else if** (OperationType.***RBracket*** == inputSymbol.**operationType**) {  
 tempSymbol = operationStack.peekLast();  
 **while** (OperationType.***LBracket*** != tempSymbol.**operationType**) {  
 **if** (operationStack.isEmpty()) {  
 **throw new** Exception(**"Wrong expression. Check it again!"**);  
 }  
 postfixNotation.add(tempSymbol);  
 operationStack.removeLast();  
 tempSymbol = operationStack.peekLast();  
 }  
 operationStack.removeLast();  
 } **else** {  
 postfixNotation.add(inputSymbol);  
 }  
 }  
 **while** (!operationStack.isEmpty()) {  
 postfixNotation.add(operationStack.pollLast());  
 }  
 **return** postfixNotation;  
 }  
  
  
 @Override  
 **public** String toString() {  
 StringBuilder builder = **new** StringBuilder();  
  
 **for**(Symbol symbol : **symbolList**) {  
 **if** (symbol.isAtom()) {  
 builder.append(symbol.**atom**.**name**).append(**" "**);  
 } **else** {  
 builder.append(symbol.**operationType**.toString()).append(**" "**);  
 }  
 }  
  
 **return** builder.toString();  
 }  
  
 **public void** printTruthTable() **throws** Exception {  
 **boolean**[][] table = getTruthTable();  
  
 System.***out***.print(**"I "**);  
 **for** (Atom atom : getAtoms()) {  
 System.***out***.print(atom.**name** + **" "**);  
 }  
 System.***out***.println(**"R"**);  
  
 **for** (**int** i = 0; i < table.**length**; i++) {  
 System.***out***.print(String.*format*(**"%2d "**, i + 1));  
 **for** (**int** j = 0; j < table[i].**length**; j++) {  
 System.***out***.print((table[i][j] ? **"t"** : **"f"**) + **" "**);  
 }  
 System.***out***.println();  
 }  
 }  
  
 **public boolean** isTautology() **throws** Exception {  
 **boolean**[][] table = getTruthTable();  
  
 **for** (**int** i = 0; i < table.**length**; i ++) {  
 **if** (!(table[i][table[i].**length** - 1])) {  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
  
 **public void** printPostfixNotation() **throws** Exception {  
 StringBuilder builder = **new** StringBuilder();  
  
 **for**(Symbol symbol : generatePostfixNotation()) {  
 **if** (symbol.isAtom()) {  
 builder.append(symbol.**atom**.**name**).append(**" "**);  
 } **else** {  
 builder.append(symbol.**operationType**.toString()).append(**" "**);  
 }  
 }  
  
 System.***out***.println(builder.toString());  
 }  
  
}

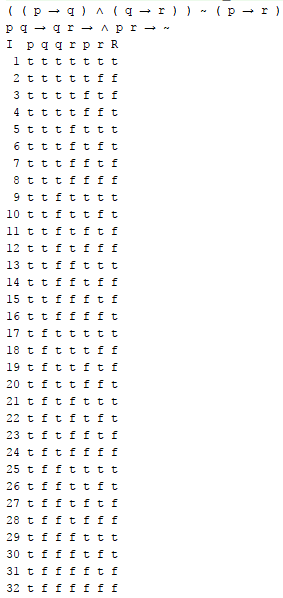
**Результат виконання програми:**

Результат виконання програми представлено на Рис.1. Як показано на рисунку було згенеровано таблицю істиності для виразу згідно з індивідуальним завданням та перевірено чи даний вираз тавтологія, в результаті даний вираз виявився не тавтологією.

**Висновок:**

При виконанні даної лабораторної роботи було реалізовано додаток для роботи з виразами Булевої логіки. Повний код програми знаходиться на репозиторії за даною URL- адресою:

<https://github.com/amukysh/EngineeringOfDataAndKnowledge.git/>



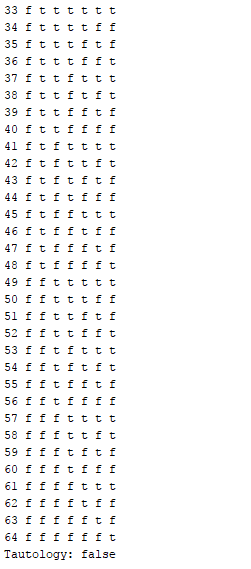


Рис.1 Результат виконання програми