МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Лабораторна робота №4

з курсу

«Інженерія даних та знань»

*на тему:*

**«Практична реалізація алгоритмів обходу дерев»**

Виконав:

ст. гр. ITIT-12

Кобак Р.З.

Прийняла:

Рибчак З.Л.

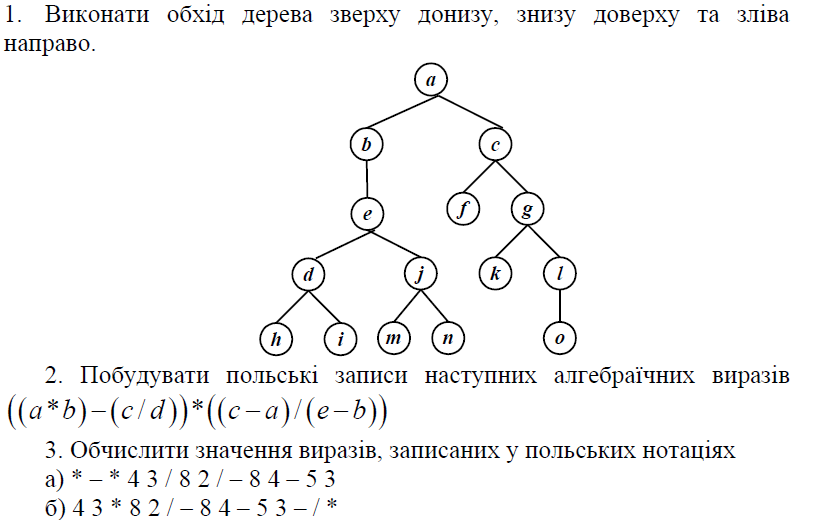
**Львів 2017**

**Мета роботи:**

Вивчення алгоритмів обходу дерев, застосування їх для запису та обчислення виразів у спеціальних (польських) нотаціях, та розробки відповідного програмного додатку

**Індивідуальне завдання:**

**Варіант 3**



**Порядок виконання:**

**Обхід дерева:**

Для обходу дерева згідно з індивідуальним завданням було реалізовано додаток, що здійсню даний обхід трьома способами та виводить результат на екран. Обхід дерева був реалізований за рекурсивними алгоритмами, що мають один суттєвий недолік, при великій глибині дерева вони можуть викликати переповнення стеку, проте при даній глибині це не суттєво.

Код програми:

**Клас BinaryTreeNode**

**public class** BinaryTreeNode<E> {  
 **public** E **value**;  
 **public** BinaryTreeNode **root**;  
 **public** BinaryTreeNode **left**;  
 **public** BinaryTreeNode **right**;  
  
 **public** BinaryTreeNode(E value) {  
 **this**.**value** = value;  
 }  
  
 **public void** setLeft(BinaryTreeNode<E> node) {  
 **left** = node;  
 **left**.**root** = **this**;  
 }  
  
 **public void** setRight(BinaryTreeNode<E> node) {  
 **right** = node;  
 **right**.**root** = **this**;  
 }  
  
 **public boolean** isLeaf() {  
 **return left** == **null** && **right** == **null**;  
 }  
  
 **public boolean** hasLeft() {  
 **return left** != **null**;  
 }  
  
 **public boolean** hasRight() {  
 **return right** != **null**;  
 }  
  
 **public** String toString() {  
 **return value**.toString();  
 }  
}

**Клас Main**

**public static void** main(String[] args) **throws** Exception {  
 System.***out***.println(PolandNotationGenerator.*toReversePolandNotation*(**"((a\*b)-(c/d))\*((c-a)/(e-b))"**));  
 System.***out***.println(PolandNotationCalculator.*calculateSimplePolandNotation*(**"\* - \* 4 3 / 8 2 / - 8 4 - 5 3"**));  
 System.***out***.println(PolandNotationCalculator.*calculateReversePolandNotation*(**"4 3 \* 8 2 / - 8 4 - 5 3 - / \*"**));  
  
 BinaryTreeNode<String> root = *generateTree*();  
 System.***out***.println(**"Left to Right"**);  
 *printLeftToRigth*(root);  
 System.***out***.println();  
  
 System.***out***.println(**"Up to down "**);  
 *printUpToDown*(root);  
 System.***out***.println();  
  
 System.***out***.println(**"Down to Up "**);  
 *printDownToUp*(root);  
 System.***out***.println();  
 }  
  
 **private static void** printLeftToRigth(BinaryTreeNode node) {  
 **if** (node.hasLeft()) {  
 *printLeftToRigth*(node.**left**);  
 }  
  
 System.***out***.print(node + **" "**);  
  
 **if** (node.hasRight()) {  
 *printLeftToRigth*(node.**right**);  
 }  
 }  
  
 **private static void** printUpToDown(BinaryTreeNode node) {  
 System.***out***.print(node + **" "**);  
  
 **if** (node.hasLeft()) {  
 *printUpToDown*(node.**left**);  
 }  
  
 **if** (node.hasRight()) {  
 *printUpToDown*(node.**right**);  
 }  
 }  
  
 **private static void** printDownToUp(BinaryTreeNode node) {  
 **if** (node.hasLeft()) {  
 *printDownToUp*(node.**left**);  
 }  
  
 **if** (node.hasRight()) {  
 *printDownToUp*(node.**right**);  
 }  
  
 System.***out***.print(node + **" "**);  
 }  
  
 **public static** BinaryTreeNode<String> generateTree() {  
 BinaryTreeNode<String> root = **new** BinaryTreeNode<>(**"a"**);  
 BinaryTreeNode<String> b = **new** BinaryTreeNode<>(**"b"**);  
 BinaryTreeNode<String> c = **new** BinaryTreeNode<>(**"c"**);  
 BinaryTreeNode<String> d = **new** BinaryTreeNode<>(**"d"**);  
 BinaryTreeNode<String> e = **new** BinaryTreeNode<>(**"e"**);  
 BinaryTreeNode<String> f = **new** BinaryTreeNode<>(**"f"**);  
 BinaryTreeNode<String> g = **new** BinaryTreeNode<>(**"g"**);  
 BinaryTreeNode<String> h = **new** BinaryTreeNode<>(**"h"**);  
 BinaryTreeNode<String> i = **new** BinaryTreeNode<>(**"i"**);  
 BinaryTreeNode<String> j = **new** BinaryTreeNode<>(**"j"**);  
 BinaryTreeNode<String> k = **new** BinaryTreeNode<>(**"k"**);  
 BinaryTreeNode<String> l = **new** BinaryTreeNode<>(**"l"**);  
 BinaryTreeNode<String> m = **new** BinaryTreeNode<>(**"m"**);  
 BinaryTreeNode<String> n = **new** BinaryTreeNode<>(**"n"**);  
 BinaryTreeNode<String> o = **new** BinaryTreeNode<>(**"o"**);  
  
 root.setLeft(b);  
 root.setRight(c);  
  
 b.setLeft(e);  
  
 e.setLeft(d);  
 e.setRight(j);  
  
 d.setLeft(h);  
 d.setRight(i);  
  
 j.setLeft(m);  
 j.setRight(n);  
  
 c.setLeft(f);  
 c.setRight(g);  
  
 g.setLeft(k);  
 g.setRight(l);  
  
 l.setLeft(o);  
  
 **return** root;  
  
 }  
}

**Результат виконання програми**

Результати виконання програми представлений на Рис.1. Як видно з рисунку був проведений обхід дерева зліва на право, знизу до верху і зверху донизу.

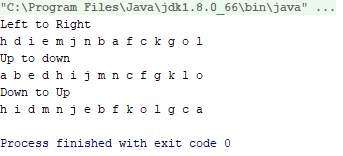


Рис.1 Результат виконання програми

**Побудова польських записів за математичним виразом**

Був розроблений додаток для парсингу математичного виразу в зворотній польський запис та пораховано значення простого польського запису для математичного виразу.

Код програми:

**Клас PolandNotationGeneration:**

**public class** PolandNotationGenerator {  
  
 **public static** String toReversePolandNotation(String input) **throws** Exception {  
 ArrayDeque<Character> operationStack = **new** ArrayDeque<>();  
 StringBuilder output = **new** StringBuilder();  
 **char** inputSymbol;  
 **char** tempSymbol;  
  
 **for** (**int** i = 0; i < input.length(); i++) {  
 inputSymbol = input.charAt(i);  
 **if** (Utils.*isOperator*(inputSymbol)) {  
 **while** (!operationStack.isEmpty()) {  
 tempSymbol = operationStack.peekLast();  
 **if** (Utils.*isOperator*(tempSymbol) && (Utils.*getOperatorPriority*(inputSymbol) <= Utils.*getOperatorPriority*(tempSymbol))) {  
 output.append(**" "**).append(tempSymbol).append(**" "**);  
 operationStack.removeLast();  
 } **else** {  
 output.append(**" "**);  
 **break**;  
 }  
 }  
 output.append(**" "**);  
 operationStack.offer(inputSymbol);  
 } **else if** (**'('** == inputSymbol) {  
 operationStack.offerLast(inputSymbol);  
 } **else if** (**')'** == inputSymbol) {  
 tempSymbol = operationStack.peekLast();  
 **while** (**'('** != tempSymbol) {  
 **if** (operationStack.isEmpty()) {  
 **throw new** Exception(**"Wrong expression. Check it again!"**);  
 }  
 output.append(**" "**).append(tempSymbol);  
 operationStack.removeLast();  
 tempSymbol = operationStack.peekLast();  
 }  
 operationStack.removeLast();  
 } **else** {  
 output.append(inputSymbol);  
 }  
 }  
 **while** (!operationStack.isEmpty()) {  
 output.append(**" "**).append(operationStack.pollLast());  
 }  
 **return** output.toString();  
 }  
  
}

**Результат виконання програми**

Результати виконання програми представлений на Рис.2. Як видно з рисунку математичний вираз було переведено в зворотній польський запис.

Прямий польський запис для математичного виразу:  
\* - \* a b / c d / - c a – e b

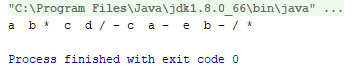


Рис.2 Результат виконання програми

**Обчислення значення виразів записаних в польських нотаціях**

Був розроблений додаток для обчислення простої і зворотної польської нотації.

Код програми:

**Клас PolandNotationCalculator:**

**public class** PolandNotationCalculator {  
  
 **public static double** calculateSimplePolandNotation(String input) **throws** Exception {  
 **double** dA = 0;  
 **double** dB = 0;  
 String temp;  
 Deque<Double> stack = **new** ArrayDeque<Double>();  
 Stack<String> st = **new** Stack<>();  
 **for** (String s : input.trim().split(**" "**)) {  
 st.push(s);  
 }  
  
 **while** (!st.isEmpty()) {  
 temp = st.pop().trim();  
 **if** (temp.length() == 1 && Utils.*isOperator*(temp.charAt(0))) {  
 **if** (stack.size() < 2) {  
 **throw new** Exception(**"Wrong data amount in stack for operation"** + temp);  
 }  
 dA = stack.pop();  
 dB = stack.pop();  
 dA = *calculate*(temp.charAt(0), dA, dB);  
 stack.push(dA);  
 } **else** {  
 dA = Double.*parseDouble*(temp);  
 stack.push(dA);  
 }  
 }  
 **return** stack.pop();  
 }  
  
 **public static double** calculateReversePolandNotation(String input) **throws** Exception {  
 **double** dA = 0;  
 **double** dB = 0;  
 String temp;  
 Deque<Double> stack = **new** ArrayDeque<Double>();  
 StringTokenizer st = **new** StringTokenizer(input);  
  
 **while**(st.hasMoreTokens()) {  
 **try** {  
 temp = st.nextToken().trim();  
 **if** (1 == temp.length() && Utils.*isOperator*(temp.charAt(0))) {  
 **if** (stack.size() < 2) {  
 **throw new** Exception(**"Wrong data amount in stack for operation"** + temp);  
 }  
 dB = stack.pop();  
 dA = stack.pop();  
 dA = *calculate*(temp.charAt(0), dA, dB);  
 stack.push(dA);  
 } **else** {  
 dA = Double.*parseDouble*(temp);  
 stack.push(dA);  
 }  
 } **catch** (Exception e) {  
 **throw new** Exception(**"Unacceptable symbol in operation"**);  
 }  
 }  
  
 **if** (stack.size() > 1) {  
 **throw new** Exception(**"Operators number and operands number are different"**);  
 }  
 **return** stack.pop();  
 }  
  
 **private static double** calculate(**char** operator, **double** dA, **double** dB) **throws** Exception {  
 **switch** (operator) {  
 **case '+'**:  
 dA += dB;  
 **break**;  
 **case '-'**:  
 dA -= dB;  
 **break**;  
 **case '/'**:  
 dA /= dB;  
 **break**;  
 **case '\*'**:  
 dA \*= dB;  
 **break**;  
 **case '%'**:  
 dA %= dB;  
 **break**;  
 **case '^'**:  
 dA = Math.*pow*(dA, dB);  
 **break**;  
 **default**:  
 **throw new** Exception(**"Unacceptable operation"** + String.*valueOf*(operator));  
 };  
 **return** dA;  
 }  
}

**Результат виконання програми**

Результати виконання програми представлений на Рис.3. Як видно з рисунку було обчислено пряму і зворотну польську нотацію, оскільки результати виконання співпадають додаток працює коректно.

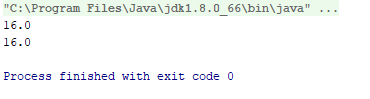


Рис.3 Результат виконання програми

**Висновок:** В результаті виконання даної лабораторної роботи було розроблено додаток для обходу бінарного дерева трьома методами, обчислення польських записів та їх генерації з математичних виразів.