Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение №3
10.03.2009г.

ТЕМА: Стек

Задача 1: Зададена е правогълна мрежа от проходими и непроходими квадратчета. По дадени две квадратчета – начално и крайно да се провери колко пътя има между тях и да се намери дължината на най-късия път. Всеки от намерените пътища да се визуализира като последователност от съставящите го квадратчета. За целта да се използва стек.

```
import java.util.Scanner;
public class Maze {

	boolean [][] maze;

	Maze(int n) {
		maze=new boolean[n][n];
	}

	void getMaze() {
		Scanner sc=new Scanner(System.in);
		for (int i=0; i<maze.length; i++)
			for (int j=0; j<maze.length; j++)
				maze[i][j]=sc.nextInt()!=0;
	}

	boolean isFree(int x, int y) {
		return maze[x][y];
	}

	void fill(int x, int y) {
		maze[x][y]=false;
	}

	void undo(int x, int y) {
		maze[x][y]=true;
	}
}
```

```
import java.util.Scanner;
import java.util.Stack;

class Pair {
	int x;
	int y;

	Pair(int x, int y) {
		this.x=x;
		this.y=y;
	}

	public String toString() {
		return "( " + x + ", " + y + " )";
	}
}

public class NetAll {

	static Maze maze;
	static int eX, eY;
	static int numberOfPath;
	static int currentLen;
	static int minLen = Integer.MAX_VALUE;
	static Stack<Pair> st = new Stack<Pair>();

	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter the length of maze: ");
		int n=sc.nextInt();
		maze=new Maze(n);
		maze.getMaze();
		int bX, bY;
		System.out.print("BEGIN X: ");
		bX=sc.nextInt();
		System.out.print("BEGIN Y: ");
		bY=sc.nextInt();
		System.out.print("END X: ");
		eX=sc.nextInt();
		System.out.print("END Y: ");
		eY=sc.nextInt();
		findAll( bX, bY);
		System.out.println("Number of path="+numberOfPath);
		System.out.println("Minimum length="+minLen);

	}
```

```
	static void findAll(int bX, int bY) {
		try {
			if(maze.isFree(bX, bY)) {
				st.push(new Pair(bX, bY));
				if(bX==eX && bY==eY)
					register();
				else {
					currentLen++;
					maze.fill(bX, bY);
					findAll(bX+1, bY);
					findAll(bX-1, bY);
					findAll(bX, bY+1);
					findAll(bX, bY-1);
					maze.undo(bX, bY);
					currentLen--;
				}
				st.pop();
			}
		}
		catch(Exception e) {}
	}

	static void register() {
		numberOfPath++;
		if(currentLen < minLen)
			minLen = currentLen;
		System.out.println("Current path is: ");
		System.out.print(st.toString());
		System.out.println();
	}
}
```

Задача 2: Да се въведе дума над азбуката { a, b, c } и да се провери дали е палиндром от вида $\alpha = \omega c \omega^{rev} / \omega \in \{a,b\}^+$.

```
import java.util.Scanner;
import java.util.Stack;
import java.util.EmptyStackException;

public class Palindrom {

	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		Stack<Character> st = new Stack<Character>();
		String input = sc.nextLine();
		char ch = '0';
```

```
                int i = 0;
                while(i < input.length() && (ch = input.charAt(i)) == 'a' || ch == 'b') {
                        st.push(ch);
                        i++;
                }
                if(i == input.length() || ch != 'c')
                        System.out.println("Word not recognized");
                else if(ch == 'c') { // continue recognizing
                        i++;
                        try {
                          while(i < input.length() && (ch = input.charAt(i))=='a' || ch=='b'){
                                  if(ch != st.pop()) {
                                          System.out.println("Word not recognized");
                                          return;
                                  } // end if
                                  i++;
                          } // end while
                        }
                        catch(EmptyStackException e) {
                                System.out.println("Word not recognized");
                                return;
                        }
                        if(i == input.length() && st.empty())
                                System.out.println("Word recognized");
                        else
                                System.out.println("Word not recognized");
                }
        }
}
```

Задача 3: Да се въведе символен низ, представляващ аритметичен израз, който съдържа цели числа, скоби и знаците за операции {+, -, *, /} и да се преобразува в обратен полски запис, след което този запис да се използва за пресметане на стойността му.

Пояснение: Постфиксен (обратен полски) запис на аритметичен израз се нарича такъв запис, при който знакът на двуаргументната операция следва след двата операнда. Необходимостта от използване на скоби при този вид запис отпада. Примери:

5*(((3+8)*(4*6))+7) инфиксен запис
5 3 8 + 4 6 * * 7 + * постфиксен (обратен полски) запис

Алгоритъм за преминаване от инфиксен към постфиксен (обратен полски) запис:
Нека ***input*** е символният низ, съдържащ израза в инфиксен запис, ***rpn*** е символен низ, който в края ще съдържа постфиксния запис, ***ptr*** е указател към поредната лексема в ***input***, ***st*** е стек. Въвеждаме следните тегла на четирите аритметични операции: умножение и деление – тегло 1, събиране и изваждане – тегло 2.

1. Ако ***input*** не е свършил, премини към 2. Иначе премини към т.3.
2. Ако ***ptr*** сочи към операнд (число или променлива), то той се добавя в края на ***rpn.*** Ако сочи лява скоба, тя се добавя в стека. Ако сочи операция, тя се добавя в стека, като предварително вади от там поред операциите, които са с тегло по-малко или равно на дадената и в реда на изваждането им ги добавя към ***rpn.*** Ако сочи дясна скоба, то от стека се изваждат всички операции до първата лява скоба и в реда на изваждането им се добавят в края на ***rpn.*** Лявата скоба се изважда от стека, но не се преписва в ***rpn.*** Указателят ***ptr*** се премества една позоция напред и се преминава към т.1.
3. Останалите в стека операции се изваждат и в същия ред се добавят в края на ***rpn.***

Алгоритъм за пресмятане на аритметичен израз, зададен в обратен полски запис:
Нека ***str*** е символният низ, който съдържа аритметичен израз, зададен в обратен полски запис, ***ptr*** е указател, който сочи към поредната лексема в него (в началото най-лявата), а ***stack*** е стек. Под лексема ще разбираме най-дългата последователност от съседни символи, която е число или знак за операция.

- Ако ***ptr*** сочи към число, добавяме това число в стека и придвижваме указателя;
- Ако ***ptr*** сочи към операция ***op***, то вадим от стека числата ***n*** и ***m*** и извършваме операцията ***m op n***, чийто резултат поместваме в стека, а указателя придвижваме напред в низа.

Когато символният низ свърши, в стека се намира числото, което е стойност на аритметичния израз, зададен от ***str.***

```
import java.util.Scanner;
import java.util.Vector;
import java.util.Stack;
import java.util.EmptyStackException;

public class ToInvPolishNotation {

    // Клас за представяне на операциите, които се записват в стека
    static enum Operation {
        BRACE ("(", 10),
        PLUS   ("+", 5),
        MINUS ("-", 5),
        MULT  ("*", 1),
        DIV     ("/", 1);

        private final String symbol;
        private final int priority;

        Operation(String s, int priority) {
            this.symbol = s;
            this.priority = priority;
        }
```

```
        int priority() {
                return priority;
        }

        public String toString() {
                return symbol;
        }
}


static char lookahead;
static Scanner sc = new Scanner(System.in);
static String input = sc.nextLine();
static int ind = 0;

public static void main(String[] args) {
        Vector<Object> v = null;
        try {
                // преобразуване в постфиксен запис
                System.out.println((v = convert()).toString());
        }
        catch(Exception e) {
                System.out.println(e.toString());
        }
        try {
                // пресмятане на аритметичния израз
                System.out.println(compute(v));
        }
        catch(Exception e) {
                System.out.println(e.toString());
        }
}

// Метод, който преобразува инфиксен запис на аритметичен израз в постфиксен
static Vector<Object> convert() throws Exception {
        Vector<Object> result = new Vector<Object>();  // Вектор за получаване на
                                                       // резултата
        Stack<Operation> st = new Stack<Operation>();  // Стек за временно пазене
                                                        // на операциите
        int val;
        lookahead=getC();
        do {  // цикъл за четене на лексемите от входа
                if (Character.isDigit(lookahead)){ // четена на лексема - число
                        val=lookahead-'0';
                        while (Character.isDigit(lookahead=getC()))
                                val=val*10+lookahead-'0';
```

```
                    result.add(new Integer(val));  // добавяне на числото във
                                                   // вектора
                }
                else {  // четене на лексема - операция
                    Operation t = null;
                    switch(lookahead) {  // добавяне на операцията в стека
                        case '(':  st.add(Operation.BRACE);
                                   break;
                        case '+': t = Operation.PLUS; insert(t, st, result);
                                   break;
                        case '-':  t = Operation.MINUS; insert(t, st, result);
                                   break;
                        case '*':  t = Operation.MULT; insert(t, st, result);
                                    break;
                        case '/':  t = Operation.DIV; insert(t, st, result);
                                   break;
                        case ')' : insertBrace(st, result);
                                   break;
                        default: throw new Exception("Error in expression");
                    }
                    lookahead=getC();
                }
        } while (lookahead != '$');
        while(!st.isEmpty())  // след край на входните лексеми, останалите в стека
                result.add(st.pop()); // операции се прехвърлят във вектора
        return result;
}


// Метод за пресмятане на аритметичен израз, зададен в обратен полски запис
static double compute (Vector<Object> v) throws Exception {
        double result = 0;
        char op;
        Stack<Number> st = new Stack<Number>(); // Стек за временно пазене
                                                // на операндите
        Object o = null;
        for(int i=0; i < v.size(); i++) {
                o = v.elementAt(i);  // четене на поредната лексема от входа
                if(o instanceof Number) // ако е число - записва се в стека
                    st.push((Number)o);
                else {
                    double operand1 = 0;
                    double operand2 = 0;
                    try {     // ако е оперция - от стека се вадят
                              // втория и
                              operand2 = ((Number)st.pop()).doubleValue();
```

```
                        // първия му операнд
                        operand1 = ((Number)st.pop()).doubleValue();
                    }
                    catch(EmptyStackException e) {
                        throw new Exception("Wrong Expression");
                    }
                    op = o.toString().charAt(0);
                    switch(op) {   // проверява се коя е операцията
                                   // и се извършва
                        case '+': result = operand1 + operand2; break;
                        case '-': result = operand1 - operand2; break;
                        case '*': result = operand1 * operand2; break;
                        case '/': result = operand1 / operand2; break;
                    }
                    st.push(new Double(result));  // резултатът се записва в стека
                }
            }
            int numb = 0;  // Когато входният поток свърши
            while(!st.isEmpty()) {  // ако в стека има точно едно число
                                    // това е резултатът от пресмятането
                result = ((Double)st.pop()).doubleValue();
                numb++;
            }
            if(numb != 1)
                throw new Exception("Wrong Expression");
            return result;
    }


    // Метод, който включва поредната прочетена операция в стека, като първо
    // прехвърля всички операции с по-висок приоритет от стека във вектора.
    static void insert(Operation t, Stack<Operation> st, Vector<Object> v) {
            Operation tt;
            while(!st.isEmpty()) {
                tt = st.peek();
                if(tt.priority() <= t.priority())
                                    v.add(st.pop());
                else break;
            }
            st.push(t);
    }


    // Методът се вика при наличие на затваряща скоба във входния поток и води до
    // прехвърляне на всички знаци за операции от стека във вектора до срещане на
    // отваряща скоба. Самата отваряща скоба не се прехвърля във вектора
```

```
	static void insertBrace(Stack<Operation> st, Vector<Object> v) {
		Operation tt;
		while(!st.isEmpty()) {
			tt = st.pop();
			if(tt != Operation.BRACE)
					v.add(tt);
			else
				break;
		}
	}

	// Метод, който чете и връща поредния символ от входа, при край връща '$'
	static char getC() {
		try {
			while(Character.isWhitespace(input.charAt(ind++)));
			return input.charAt(ind-1);
		}
		catch (Exception e) {
			return '$';
		}
	}
}
```

Задача 4: Да се напише програма, която прочита аритметичен израз, проверява дали е правилен и пресмята стойността му. Изразът може да съдържа операциите събиране, изваждане, умножение и деление, скоби и операнди – цели числа.
Следната граматика поражда такъв израз:

$\Gamma = <\{ +, -, *, /, (,), numb\}, \{E, T, F\}, E,$
$\{E \rightarrow E + T| E - T| T,$
$T \rightarrow T * F| T / F| F,$
$F \rightarrow (E)| numb \} >$

Същата граматика след елиминиране на лявата рекурсия и лява факторизация:

$\Gamma' = <\{ +, -, *, /, (,), numb\}, \{E, T, F, E', T'\}, E,$
$\{E \rightarrow TE',$
$E' \rightarrow +TE'| -TE'| \varepsilon,$
$T \rightarrow FT',$
$T' \rightarrow *FT'| /FT'| \varepsilon,$
$F \rightarrow (E)| numb \} >$

```
import java.util.Scanner;
public class Expression {

        static char lookahead;
        static Scanner sc = new Scanner(System.in);
        static String input = sc.nextLine();
        static int ind = 0;

        public static void main(String[] args) {
                lookahead = getC();
                try {
                double val = expr();
                if (lookahead == '$')
                        System.out.println(val);
                else
                        System.out.println("Wrong Expression");
        }
                catch (ExpressionError e) {
                        System.out.println("Wrong Expression");
                }
         }

         static double expr() throws ExpressionError {
                double val = term();
                while (lookahead == '+' || lookahead == '-')
                        if (lookahead=='+') {
                                lookahead = getC();
                                val += term();
                        }
                        else {
                                lookahead = getC();
                                val -= term();
                        }
                        return val;
          }

          static double term() throws ExpressionError {
                double val = factor();
                while (lookahead == '*' || lookahead == '/')
                        if (lookahead == '*') {
                                lookahead = getC();
                                val *= factor();
                        }
                         else {
                                 lookahead = getC();
                                val /= factor();
```

```
                }
                return val;
        }

        static double factor() throws ExpressionError {
                double val;
                if (lookahead == '(') {
                        lookahead = getC();
                        val = expr();
                        if (lookahead != ')')
                                throw new ExpressionError();
                        lookahead = getC();
                        return val;
                }
                else if (Character.isDigit(lookahead)){
                         val = lookahead-'0';
                        while (Character.isDigit(lookahead = getC()))
                                val = val*10+lookahead-'0';
                        return val;
                }
                else throw new ExpressionError();
        }

        static char getC() {
                try {
                        while(Character.isWhitespace(input.charAt(ind++)));
                        return input.charAt(ind-1);
                }
                catch (Exception e) {
                        return '$';
                }
        }
}
```

Домашно №2 със срок на предаване 20.03.2009г.

Задача 1: Да се напише програма , която проверява дали дума над азбуката { (,) } е правилен скобов израз като използва стек.

Задача 2: Да се напише програма, която проверява дали дума над азбуката { 0 , 1} принадлежи на езика:

L={$\alpha \in$ {0,1}*/броят на единиците е равен на удвоения брой на нулите}