자료구조 실습 보고서
[제 8 주] 스택 : 수식계산

제출일 : 2014 - 04 - 28

201402395 이승희

1. 프로그램 설명서
   1) 주요 알고리즘 / 자료구조 / 기타
      : 키도브에서 한 번에 하나의 문자열을 입력 받는다. 문자열은 infix 수식을 나타내며, 수식 표현에 오류는 없다고 가정한다. 입력의 종료 조건은 '!'키를 치면 프로그램을 종료한다. 매번 하나의 수식을 입력받을 때마다 수식을 Postfix 형태로 변환하며 Postfix 수식을 계산한다.
   2) 함수 설명서
      (1) AppContoller
         - public void evalExpression() // 주어진 infix 수식을 계산한다.
         - Public void run() // 프로그램의 실질적인 실행을 위한 함수. 실질적인 main() 함수
      (2) AppView
         - public String inputString() // 문자열을 입력받는다.
         - public String inputExpression() // 수식을 입력받는다.
         - Public void outputResult(double aValue) // 최종값을 출력한다.
         - Public void outputPostfix(String aPostfix) // Postfix 를 출력한다.
      (3) Calculate
         - public void setInfix(String anInfix) // 입력받은 infix 를 Calculate 의 infix 에 저장
         - public String infix() // 저장 되어 있는 infix 를 전달
         - public String postfix() // 저장 되어 있는 postfix 를 전달
         - public Boolean infixToPostfix() // Infix 를 postfix 로 변경
         - public double evalPostfix() // postfix 로 변경된 수식을 계산
         - public void showOStackAll() // 임시로 스택의 상태를 확인 하기 위해 사용하는 함수이며, Operation 스택 내부에 있는 모든 값을 출력
         - public void showVStackAll() // 임시로 스택의 상태를 호가인 하기 위해 사용하는 함수로, value 스택 내부에 있는 모든 값을 출력
         - private Boolean isDigit(char aToken) // 전달 받은 Token 이 숫자인지 확인하여 숫자일 경우 true 를 반환
         - private int inComingPrecedence(char aToken) // 앞으로 실행되어야 하는 연산자가 가져야 하는 우선 순위 값을 반환
         - private int inStackPrecedence(char aToken) // 스택 내부에 존재하는 연산자의 우선 순위 값을 반환
   3) 종합 설명서
      : infix 수식을 postfix 수식으로 변환하는 프로그램을 stack 의 개념을 도입하여 프로그래밍한다.

2. 프로그램 장단점 분석

   : 스택의 개념을 이용하여, 연산이 끝난 operator 는 바로 pop 하여 출력하기 때문에 infix 를 postfix 로 바꾸는 과정을 모두 확인할 수 있다.

3. 실행 결과 분석

   1) 입력과 출력

```
:: 프로그램을 시작합니다. ::
[수식 입력을 시작합니다.]
> 수식을 입력하시오 : (3+6-7)^(8-1*5)^(9/4)$
[Infix를 Postfix로]
OStack : (
OStack : ( +
OStack : ( -
OStack :
OStack : ^
OStack : ^ (
OStack : ^ ( -
OStack : ^ ( - *
OStack : ^
OStack : ^ ^
OStack : ^ ^ (
OStack : ^ ^ ( /
OStack : ^ ^
OStack : $

[Postfix] 36+7-815*-94/^^$
VStack : 3.0
VStack : 3.0 6.0
VStack : 9.0
VStack : 9.0 7.0
VStack : 2.0
VStack : 2.0 8.0
VStack : 2.0 8.0 1.0
VStack : 2.0 8.0 1.0 5.0
VStack : 2.0 8.0 5.0
VStack : 2.0 3.0
VStack : 2.0 3.0 9.0
VStack : 2.0 3.0 9.0 4.0
VStack : 2.0 3.0 2.25
VStack : 2.0 27.0
VStack : 1.34217728E8
[최종값] 1.34217728E8

> 수식을 입력하시오 : !
[수식 입력을 종료합니다.]
:: 프로그램을 종료합니다. ::
```

   2) 결과분석

   : 수식 (3+6-7)^(8-1*5)^(9/4)$ 을 입력하면 각 과정에 따라 operator 의 stack 과 value 의 stack 을 각각 출력하며, '!'입력시 프로그램을 종료한다.

4. 소스코드

1) AppController

```
public class AppController {
    private AppView _appView;
    private Calculate _calculate;

    public AppController() {
        this._appView = new AppView();
        this._calculate = new Calculate();
    }

    public void evalExpression() {
        double finalValue;
        this.showMessage(MessageID.Notice_InfixToPostfix);
        if (this._calculate.infixToPostfix()) {
            this._appView.outputPostfix(this._calculate.postfix());
            finalValue = this._calculate.evalPostfix();
            this._appView.outputResult(finalValue);
        } else
            this.showMessage(MessageID.Error_Input);
    }

    public void run() {
        this.showMessage(MessageID.Notice_StartProgram);
        this.showMessage(MessageID.Notice_StartMenu);
        String input = this._appView.inputExpression();
        while (input.charAt(0) != '!') {
            this._calculate.setInfix(input);
            this.evalExpression();
            input = this._appView.inputExpression();
        }
        this.showMessage(MessageID.Notice_EndMenu);
        this.showMessage(MessageID.Notice_EndProgram);

    }

    private void showMessage(MessageID aMessageID) {
        switch (aMessageID) {
        case Notice_StartProgram:
            System.out.println(":: 프로그램을 시작합니다. ::");
            break;
        case Notice_EndProgram:
            System.out.println(":: 프로그램을 종료합니다. ::");
            break;
        case Notice_StartMenu:
            System.out.println("[수식 입력을 시작합니다.]");
            break;
        case Notice_EndMenu:
            System.out.println("[수식 입력을 종료합니다.]");
            break;
        case Notice_InfixToPostfix:
            System.out.println("[Infix를 Postfix로]");
            break;
        default:
            System.out.println("잘못된 입력입니다.");
        }
    }
}
```

2) AppView

```
import java.util.*;

public class AppView {
    private Scanner _scanner;

    public AppView() {
        this._scanner = new Scanner(System.in);
    }

    public String inputString() {
        return this._scanner.nextLine();
    }

    public String inputExpression() {
        System.out.print("> 수식을 입력하시오 : ");
        return this.inputString();
    }

    public void outputResult(double aValue) {
        System.out.println("[최종값] " + aValue);
        System.out.println();
    }

    public void outputPostfix(String aPostfix) {
        System.out.println();
        System.out.println("[Postfix] " + aPostfix);
    }

}
```

3) DS1_08_201402395_이승희

```
public class DS1_08_201402395_이승희 {

    public static void main(String[] args) {
        AppController appController = new AppController();
        appController.run();

    }

}
```

4) MessageID

```
public enum MessageID {
    Notice_StartProgram,
    Notice_EndProgram,
    Notice_StartMenu,
    Notice_EndMenu,
    Notice_InfixToPostfix,
    Error_Input,

}
```

5) ArrayList

```
public class ArrayList<T> implements Stack<T> {

    private static final int DEFAULT_MAX_STACK_SIZE = 5;
    private int _maxSize;
    private int _top;
    private T[] _elements;

    @SuppressWarnings("unchecked")
    public ArrayList() {
        this._maxSize = DEFAULT_MAX_STACK_SIZE;
        this._top = -1;
        this._elements = (T[]) new Object[this.DEFAULT_MAX_STACK_SIZE];
    }

    public ArrayList(int maxSize) {
        this._maxSize = maxSize;
        this._top = -1;
        this._elements = (T[]) new Object[this.DEFAULT_MAX_STACK_SIZE];
    }

    public boolean isEmpty() {
        return this._top == -1;
    }

    public boolean isFull() {
        return this._top == DEFAULT_MAX_STACK_SIZE-1;
    }

    public int size() {
        return this._top+1;
    }

    @Override
    public boolean push(T anElement) {
        if (this.isFull())
            return false;
        else {
            this._top++;
            this._elements[this._top]=anElement;
            return true;

        }
    }

    @Override
    public T pop() {
        if (this.isEmpty())
            return null;
        else {
            this._top--;
            return this._elements[this._top+1];
        }

    }

    @Override
    public T peek() {
        if (this.isEmpty())
            return null;
        else {
            return this._elements[this._top];
        }
    }

    public void clear() {
        this._top = -1;
        this._elements = null;
    }

    public T elementAt(int order) {
        return this._elements[order];
    }

}
```

6) Calculation

```
public class Calculate {
    private ArrayList<Character> _oStack;
    private ArrayList<Double> _vStack;
    private char[] _infix;
    private char[] _postfix;

    public Calculate() {
        this._infix = null;
        this._oStack = null;
        this._postfix = null;
        this._vStack = null;
    }

    public void setInfix(String anInfix) {
        this._infix = anInfix.toCharArray();
    }

    public String infix() {
        if (this._infix != null)
            return String.valueOf(this._infix);
        else
            return null;
    }

    public String postfix() {
        if (this._postfix != null)
            return String.valueOf(this._postfix);
        else
            return null;
    }

    private boolean isDigit(char aToken) {
        return (aToken >= '0' && aToken <= '9');
    }

    public void showOStackAll() {
        System.out.print("OStack : ");
        for (int i = 0; i < this._oStack.size(); i++)
            System.out.print(this._oStack.elementAt(i) + " ");
        System.out.println();
    }

    public void showOStackAll() {
        System.out.print("OStack : ");
        for (int i = 0; i < this._oStack.size(); i++)
            System.out.print(this._oStack.elementAt(i) + " ");
        System.out.println();
    }

    public void showVStackAll() {
        System.out.print("VStack : ");
        for (int i = 0; i < this._vStack.size(); i++)
            System.out.print(this._vStack.elementAt(i) + " ");
        System.out.println();
    }
```

```
public boolean infixToPostfix() {
    int i = 0, p = 0;
    char currentToken, poppedToken, topToken;

    this._oStack = new ArrayList<Character>(this._infix.length);
    this._postfix = new char[this._infix.length];

    while (i < this._infix.length) {
        currentToken = this._infix[i++];
        if (this.isDigit(currentToken))
            this._postfix[p++] = currentToken;
        else {
            if (currentToken == ')') {
                if (!this._oStack.isEmpty())
                    poppedToken = (char) this._oStack.pop();
                else
                    return false;
                while (poppedToken != '(') {
                    this._postfix[p++] = poppedToken;
                    if (!this._oStack.isEmpty())
                        poppedToken = (char) this._oStack.pop();
                    else
                        return false;
                }
                this.showOStackAll();
            } else {
                int inComingP = this.inComingPrecedence(currentToken);
                if (!this._oStack.isEmpty()) {
                    topToken = (char) this._oStack.peek();
                    while (this.inStackPrecedence(topToken) >= inComingP) {
                        poppedToken = (char) this._oStack.pop();
                        this._postfix[p++] = poppedToken;
                        if (!this._oStack.isEmpty())
                            topToken = (char) this._oStack.peek();
                        else
                            break;

                    }
                }
                this._oStack.push(currentToken);
                this.showOStackAll();
            }
        }

    }
```

```
    while (!this._oStack.isEmpty()) {
        poppedToken = (char) this._oStack.pop();
        this._postfix[p++] = poppedToken;
    }
    return true;
}

public double evalPostfix() {
    int p;
    char curToken;
    this._vStack = new ArrayList<Double>(this._infix.length);

    p = 0;
    while (p < this._postfix.length) {
        curToken = this._postfix[p++];
        if (this.isDigit(curToken)) {
            this._vStack.push(Double.parseDouble(String.valueOf(curToken)));
            this.showVStackAll();
        } else {
            double operand1 = this._vStack.pop();
            double result = 1;

            if (curToken == '+') {
                double operand2 = this._vStack.pop();
                result = operand1 + operand2;
            } else if (curToken == '-') {
                double operand2 = this._vStack.pop();
                result = operand2 - operand1;
            } else if (curToken == '*') {
                double operand2 = this._vStack.pop();
                result = operand1 * operand2;
            } else if (curToken == '/') {
                double operand2 = this._vStack.pop();
                result = operand2 / operand1;
            } else if (curToken == '%') {
                double operand2 = this._vStack.pop();
                result = operand2 % operand1;
            } else if (curToken == '^') {
                double operand2 = this._vStack.pop();
                for (int i = 0; i < operand1; i++)
                    result *= operand2;
            } else {
                this._vStack.push(operand1);
                return this._vStack.peek();
            }
            this._vStack.push(result);
            this.showVStackAll();
        }
    }
    return this._vStack.peek();
}

private int inComingPrecedence(char aToken) {
    if (aToken == '+')
        return 12;
    else if (aToken == '-')
        return 12;
    else if (aToken == '(')
        return 20;
    else if (aToken == ')')
        return 19;
    else if (aToken == '*')
        return 13;
    else if (aToken == '/')
        return 13;
    else if (aToken == '%')
        return 13;
    else if (aToken == '^')
        return 17;
    else if (aToken == '$')
        return 0;
    else
        return -1;
}
```

```
    private int inStackPrecedence(char aToken) {
        if (aToken == '+')
            return 12;
        else if (aToken == '-')
            return 12;
        else if (aToken == '(')
            return 0;
        else if (aToken == ')')
            return 19;
        else if (aToken == '*')
            return 13;
        else if (aToken == '/')
            return 13;
        else if (aToken == '%')
            return 13;
        else if (aToken == '^')
            return 16;
        else if (aToken == '$')
            return 0;
        else
            return -1;
    }

}
```

7) Stack

```
public interface Stack<T> {

    public boolean push(T anElement);

    public T pop();

    public T peek();

}
```

5. 생각해볼 점
   1) postfix 수식에서 연산값 token 들은 아직 정수 값으로 변환되지 않은 문자상태이다. 연산을 위해서는 이 token 들을 정수값으로 변환해야 한다. 그러므로 postfix 에서 operand 를 얻자마자 바로 정수값으로 바꾸기로한다. 정수로 바꾸고 해야할 일은?
      : showVStackAll()
   2) 계산을 위한 스택인 ValueStack 은?
      스택에 들어가는 값은 실수값이어야한다. 즉, ValueStack 의 T 의 타입은 double 이다.
      : 나누기 계산을 해야하기 때문에 double 로 선언한다.
   3) Token 이 연산자 일때, 스택에 있는 값을 꺼내어 연산을 실행하고 연산이 끝나면 계산이 끝난 값을 다시 스택에 넣는다. 연산이 끝나고 난 뒤 계산 결과값은 스택의 top 에 있는 값이다. 왜그럴까?
      : 연산이 끝난 후 항상 vStack 에 push 되기 때문이다.

4) Postfix 연산 중에 일어나는 스택 변환 과정을 그린 뒤 보고서에 첨부하세요.