# 자료구조 및 실습
# 보고서

[제05주] DS_05_201502273_김현종

2018.04.12.

201502273/김현종

# 1.내용

```
public class Stack {
    private String[] stack;
    private int size;
    private int maxSize;
    private final static int DEFAULT_MAX_SIZE = 5;

    Stack() {
        //TODO
        this(Stack.DEFAULT_MAX_SIZE);     //Stack(int) 호출
    }

    /*
     * Stack(int )생성자
     * 받은 값으로 스택의 최댓값을 지정하고
     * 스택으로 사용할 String Array를 선언한다.
     * 현재 size는 -1로 초기화한다.
     */
    Stack(int maxSize){
        this.maxSize = maxSize;
        this.stack = new String[this.maxSize];
        size = -1;
    }

    /*
     * 스택에 값을 Push하는 메소드
     * 현재 Stack이 꽉 차있는지 확인한 뒤 꽉 찼으면 사이즈를 늘리고 자신을 다시 호출한다.
     * 꽉 차지 않았으면 현재 Stack의 size를 1 올리고 받은 string값을 peek에 넣는다.
     * 그리고 true를 반환한다.
     */
    public boolean push(String string) {
        //TODO
        if(!isFull()){
                this.size++;
        this.stack[this.size] = string;
        return true;
        }else{
                this.resize();
                return this.push(string);
        }
    }

    /*
     * Stack의 pop메소드
     * 반환할 값을 현재 Stack의 peek에서 가져온 후 Stack의 현재 size를 1 줄인다.
     * 그리고 삭제한 값을 반환한다.
     */
    public String pop() {
        //TODO
        String ret = this.stack[this.size];
        this.size--;
        return ret;
    }

    /*
     * Stack의 peek값을 반환하는 함수
     * 현재 Stack의 맨 위에 있는 값을 반환한다.
     */
    public String peek() {
        //TODO
        if(!isEmpty())        return this.stack[this.size];
        else return "";
    }

    /*
     * 현재 Stack의 size값이 -1이면 true를 반환한다.
     */
    public boolean isEmpty() {
        //TODO
        return this.size == -1;
```

```
    }

    /*
     * 현재 Stack의 size + 1이 Stack의 maxSize와 같으면  true를 반환한다.
     */
    public boolean isFull() {
        //TODO
         return this.size+1 == this.maxSize;
    }

    /*
     * Stack의 크기를 재조정하는 함수
     * 현재 maxSize의 두 배의 크기를 갖는 어레이를 만들고
     * 현재 stack이라는 어레이의 값을 복사한 뒤 현재값은 버리고 새로만든값을 취한다.
     * maxSize는 두배로 한다.
     */
    private void resize() {
        String[] newStack = new String[maxSize * 2];
        System.arraycopy(stack, 0, newStack, 0, maxSize); // 메모리를 복사합니다.
        stack = newStack;
        maxSize *= 2;
    }
}
```

Stack.java

```
public class Postfix {
    private String infix;
    private StringBuilder postfix;

    /*
     * Postfix의 생성자
     * infix를 받아온 값으로 초기화하고
     * postfix는 새로운 StringBuilder 객체를 만들어 초기화하고
     * infix를 postfix로 만들기 위해 infixToPostfix를 호출한다.
     */
    public Postfix(String string) {
        infix = string;
        postfix = new StringBuilder();
        infixToPostfix(0);
    }

    /*
     * infix로 작성된 수식을 postfix로 바꾸는 메소드
     * 받아온 인덱스값부터 그 이후를 처리한다.
     */
    private int infixToPostfix(int index) {
        StringBuilder num = new StringBuilder();      //숫자를 임시로 저장할 StringBuilder
        Stack operationStack = new Stack();//연산자를 저장할 Stack
        for (int i = index; i < infix.toCharArray().length; ++i) {//받아온 인덱스부터 infix
스트링의 끝까지
            char ch = infix.charAt(i);          //i번째 자리에 있는 Char
            switch (ch) {
                case '(':   //i번째 char가 (일 경우
                    i = infixToPostfix(i+1);//(이후부터 recursive하게 다시 처리한 뒤
                    //해당 괄호가 끝난 인덱스를 받아온다.
                    break;
                case ')'://i번째 char가 )일 경우
                    makePostfix(num.toString());        //현재까지 받은 num에 든
숫자값(Operand)을 postfix에 넣어준다.
                    while (!operationStack.isEmpty()) {//연산자 스택이 빌때까지
                        makePostfix(operationStack.pop());//연산자 스택에 남은 연산자를
pop하여 postfix에 넣어준다.
                    }
                    return i;         //)가 나타난 인덱스를 반환한다.
                case '*'://i번째 Char가 *일 경우
                    //TODO
                  makePostfix(num.toString());//현재까지 받은 num에 든 숫자값(Operand)을
postfix에 넣어준다.
```

```
                    num = new StringBuilder();//num을 비워준다(다시 초기화)
                    if(!operationStack.isEmpty()){//연산자 스택이 비지 않았으면
                            if(operationStack.peek().equals("/") ||
operationStack.peek().equals("*")){//현재 연산자 스택의 peek값이 /, *이면
                                    makePostfix(operationStack.pop());//연산자 스택의
peek값을 pop하여 postfix에 넣어준다.
                            }
                    }
                    operationStack.push("*");//연산자 스택에 *을 넣어준다.
                    break;

                case '/'://i번째 Char가 /일 경우
                    //TODO
                    makePostfix(num.toString());//현재까지 받은 num에 든 숫자값(Operand)을
postfix에 넣어준다.
                    num = new StringBuilder();//num을 비워준다(다시 초기화)
                    if(!operationStack.isEmpty()){//연산자 스택이 비지 않았으면
                            if(operationStack.peek().equals("*") ||
operationStack.peek().equals("/")){//현재 연산자 스택의 peek값이 /, *이면
                                    makePostfix(operationStack.pop());//연산자 스택의
peek값을 pop하여 postfix에 넣어준다.
                            }
                    }
                    operationStack.push("/");//연산자 스택에 /을 넣어준다.
                    break;

                case '+'://i번째 Char가 +일 경우
                    makePostfix(num.toString());//현재까지 받은 num에 든 숫자값(Operand)을
postfix에 넣어준다.
                    num = new StringBuilder();//num을 비워준다(다시 초기화)
                    while (!operationStack.isEmpty()) {//연산자 스택이 빌때까지(우선순위가
가장 낮기때문에)
                        makePostfix(operationStack.pop());//연산자 스택을 pop하여 postfix에
넣어준다.
                    }
                    operationStack.push("+");//연산자 스택에 +을 넣어준다.
                    break;
                case '-'://i번째 Char가 -일 경우
                    makePostfix(num.toString());//현재까지 받은 num에 든 숫자값(Operand)을
postfix에 넣어준다.
                    num = new StringBuilder();//num을 비워준다(다시 초기화)
                    while (!operationStack.isEmpty()) {//연산자 스택이 빌때까지(우선순위가
가장 낮기때문에)
                        makePostfix(operationStack.pop());//연산자 스택을 pop하여 postfix에
넣어준다.
                    }
                    operationStack.push("-");//연산자 스택에 -을 넣어준다.
                    break;
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9'://i번째 Char가 숫자일 경우
                    num.append(ch);//num에 append한다.(이어 붙힌다.)
                    break;
                case ' ':
                case '\t':
                case '\n'://i번째 Char가 ' ', '\t', '\n'일 경우
                    break;//아무것도 하지 않고 끝낸다.
                default://i번째 Char가 위에것 이외의 것일 경우
                    System.out.println("[ 올바르지 않은 입력입니다. : " + ch + " ]");//에러를
출력한다.
                    break;
            }
        }//반복문이 모두 끝나고
        makePostfix(num.toString());//현재 num에 들어있는 숫자를 postfix에 먼저 넣어주고
        while (!operationStack.isEmpty()) {//연산자 스택이 빌때까지
```

```
                makePostfix(operationStack.pop());//연산자 스택에서 pop하여 postfix에 넣어준다.
            }
            return infix.toCharArray().length;//infix의 길이, 즉 마지막 인덱스+1을 반환한다.
        }

        private void makePostfix(String string) {
            if (!"".equals(string)) {//입력값이 ""이 아닐경우
                postfix.append(string).append(" ");//입력값을 postfix에 append하고 빈칸도 하나
append한다.
            }
        }

        /*
         * 제작된 postfix를 실제로 계산하는 메소드
         *
         */
        public double calculate() {
            Stack valueStack = new Stack();      //valueStack을 만듬
            StringBuilder num = new StringBuilder();//숫자가 들어갈 num이라는 StringBuilder를
생성함.

            double result;//결과값이 들어갈 변수
            for (int i = 0; i < postfix.toString().toCharArray().length; ++i) {//0부터 postfix의
끝까지
                char ch = postfix.charAt(i);//i번째 char
                switch (ch) {
                    case '*': {//i번째 char가 *인 경우
                        //TODO
                      String b = valueStack.pop();
                        String a = valueStack.pop();//valueStack에서 두개의 값을 pop하여
                        a = Double.toString(Double.parseDouble(a) * Double.parseDouble(b));
                        //해당 값을 곱하고 결과값을 다시 valueStack에 push한다.
                        valueStack.push(a);
                        break;
                    }
                    case '/': {//i번째 char가 /인 경우
                        //TODO
                      String b = valueStack.pop();
                        String a = valueStack.pop();//valueStack에서 두개의 값을 pop하여
                        a = Double.toString(Double.parseDouble(a) / Double.parseDouble(b));
                        //해당 값을 나누고 결과값을 다시 valueStack에 push한다.
                        valueStack.push(a);
                        break;
                    }
                    case '+': {//i번째 char가 +인 경우
                        String b = valueStack.pop();
                        String a = valueStack.pop();//valueStack에서 두개의 값을 pop하여
                        a = Double.toString(Double.parseDouble(a) + Double.parseDouble(b));
                        //해당 값을 더하고 결과값을 다시 valueStack에 push한다.
                        valueStack.push(a);
                        break;
                    }
                    case '-': {//i번째 char가 +인 경우
                        String b = valueStack.pop();
                        String a = valueStack.pop();//valueStack에서 두개의 값을 pop하여
                        a = Double.toString(Double.parseDouble(a) - Double.parseDouble(b));
                        //해당 값을 빼고 결과값을 다시 valueStack에 push한다.
                        valueStack.push(a);
                        break;
                    }
                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
                    case '7':
                    case '8':
                    case '9':
                    case '.'://i번째 값이 숫자인 경우 num에 append해준다.
                        num.append(ch);
```

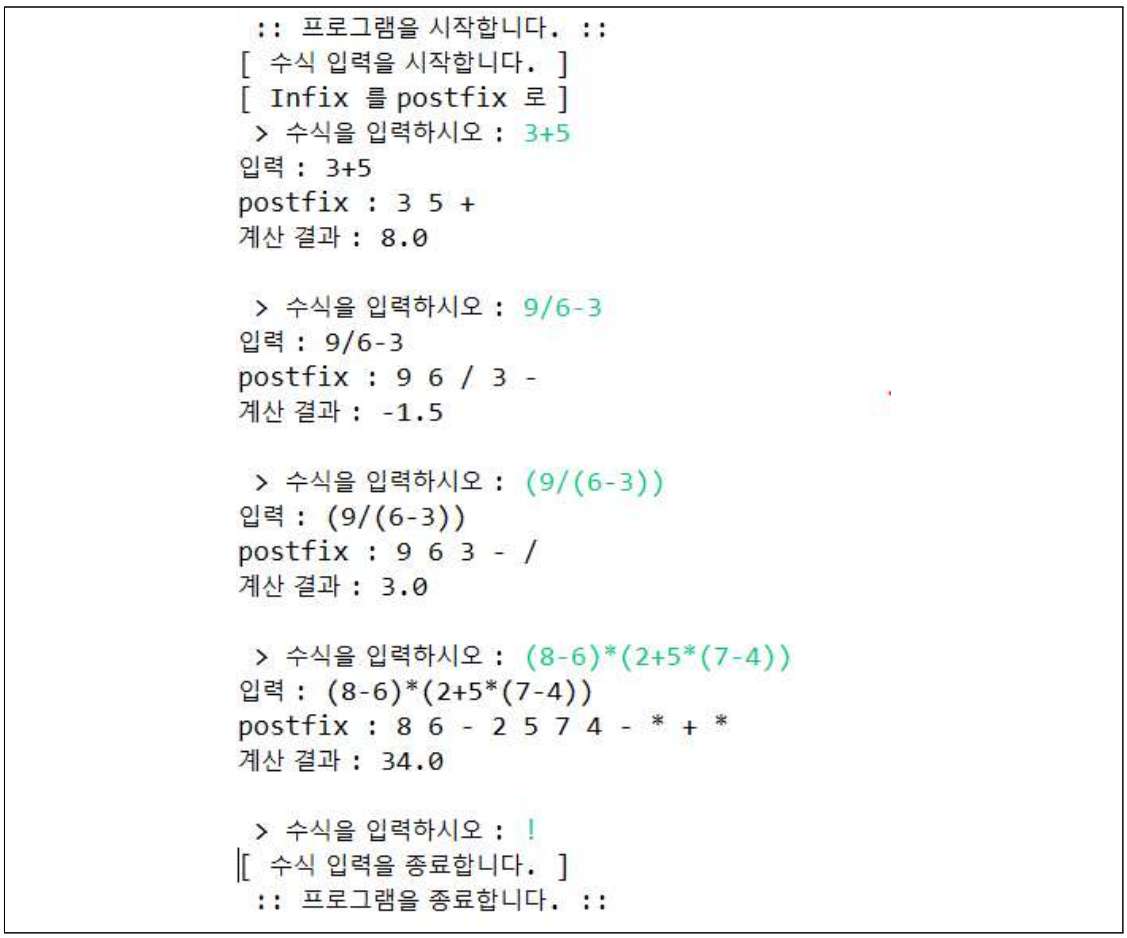
```
                    break;
                case ' '://i번째 값이 ' '인 경우
                    if (!num.toString().equals("")) {//num이 ""가 아닐경우
                        valueStack.push(num.toString());//num에 있는 숫자값을
valueStack에 push해준다.
                        num = new StringBuilder();//num을 비운다.(초기화 한다.)
                    }
                    break;
                default://위의 값 이외의 값이 i번째 값이면 아무것도 하지 않는다.
                    break;
            }
        }

        result = Double.valueOf(valueStack.pop());//valueStack의 마지막 값을 출력함
        return result;//결과값을 반환한다.
    }

    /*
     * infix의 getter
     */
    public String getInfix() {
        return infix;
    }

    /*
     * postfix의 getter
     */
    public String getPostfix() {
        return postfix.toString();
    }
}
```

Postfix.java

```
import java.util.Scanner;

public class MainClass_05_201502273 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println(" :: 프로그램을 시작합니다. :: ");
        System.out.println("[ 수식 입력을 시작합니다. ]");
        System.out.println("[ Infix 를 postfix 로 ]");
        String string;
        while (true) {

            System.out.print(" > 수식을 입력하시오 : ");
            string = scanner.nextLine();
            if ("!".equals(string))
                break;

            Postfix postfix = new Postfix(string);

            System.out.println("입력 : " + postfix.getInfix());
            System.out.println("postfix : " + postfix.getPostfix());

            System.out.println("계산 결과 : " + postfix.calculate()+"\n");

        }
        System.out.println("[ 수식 입력을 종료합니다. ]");
        System.out.println(" :: 프로그램을 종료합니다. :: ");
    }
}
```

MainClass_05_201502273.java

## 2.결과




infix 수식을 입력받아 먼저 postfix 수식으로 변환하는 것을 볼 수 있다. 그리고 만들어진 postfix 수식에서 하나씩 값을 꺼내며 연산을 한 뒤 연산결과를 보여주는 것을 볼 수 있다.

### 깨달은 점 및 결론

수식에서 여는 괄호를 만난 경우 닫는 괄호를 만날 때까지의 수식을 Recursive하게 처리하는 것을 알 수 있었다. 해당 실습에서는 이론수업에서 필요했던 우선순위 테이블 대신에 괄호는 Recursive하게 처리하며 연산자는 +, -, *, /의 사칙연산만 구현을 했기 때문에 해당 연산자 Set에서 우선순위가 가장 낮은 +와 -가 나왔을 때는 현재 연산자 Stack을 모두 비우며 postfix stack에 넣어주고 +혹은 -를 연산자 Stack에 넣어주는 것을 알 수 있다. 또한 연산자가 *, /가 나오면 연산자 Stack의 peek값이 *혹은 /인지 확인을 하고 만약 우선순위가 같은 *혹은 /가 peek에 있으면 연산자 Stack에서 하나 pop하여 postfix Stack에 push해주고 수식에 있던 *, /를 연산자 스택에 넣어준다.
해당 시스템은 사칙연산을 수행할 때 좌측을 우선적으로 연산하는 시스템이기 때문에 *가 들어왔을 때 연산자 Stack에 peek값이 똑같은 *여도 pop하여 postfix Stack에 push해주지만 우측을 우선적으로 연산하는 시스템의 경우 peek를 확인하여 같은 우선순위의 연산자가 나오면 pop하지 않고 연산자 Stack에 받은 연산자를 push해주면 될 것이다.