Computer Science and technology

**The Developments of a Calculator**

**by Java and YACC**

Major： Software Engineering

Class ： Class 4

ID NO.： 201312172063

Name： Chen Wanli (陈万礼)

[Contents](app:ds:contents)

[Contents I](#_Toc389751761)

[Development of a Calculator by Java 1](#_Toc389751762)

[1. Introduction 1](#_Toc389751763)

[2. System Environment 1](#_Toc389751764)

[3. Design Instruction 1](#_Toc389751765)

[4. Codes of core technologies 2](#_Toc389751766)

[5. Tests of the calculator 5](#_Toc389751767)

[5.1 Unit tests 5](#_Toc389751768)

[5.2 Integration tests 5](#_Toc389751769)

[6. GUI of the calculator 6](#_Toc389751770)

[7. Summary 6](#_Toc389751771)

[Appendix: Development of a Calculator by YACC 7](#_Toc389751772)

[System Environment 7](#_Toc389751773)

[Codes of YACC 7](#_Toc389751774)

[Compile the codes of YACC 8](#_Toc389751775)

[Test the generated file 8](#_Toc389751776)

Development of a Calculator by Java

# Introduction

This program mainly implemented a calculator with Java, which could process mathematical operators, including +,-，\*,/,^,( and ) . The main methods used in the program were stack and Reverse Polish Expression.

# System Environment

1. **Hardware Environment:** A PC , parameters of which are listed as below.

CPU: Intel Core i7 4500U, 1.8GHz.

Memory Capacity: 3GB

1. **Software Environment**: Windows 7 (32bit) , JDK1.6.0\_10, MyEclipse8.5.

# Design Instruction

The calculator mainly consists of three parts: GUI System, Reverse Polish Expression System (RPE System), Mathematical Calculation System( MC System).



1. GUI System mainly helps the calculator communicate with the user.
2. RPE System mainly analyzes the infix expression, and convert the infix expression into postfix expression ,which is also called Reverse Polish Expression.
3. MC System mainly calculates the result with a stack according to the Reverse Polish Expression.

In the program, I have mainly written two classes and one interface. And the Class Diagram is as below.



In this diagram, the function ” *Calculator.infix2suffix()*” mainly realizes the role of RPE System; and the function “*Calculator.eval()*” mainly realizes the role of MC System.

# Codes of core technologies

This program has two mainly technologies: Reverse Polish Expression and calculating stack.

1. **Codes of Reverse Polish Expression**

This party is mainly used to convert the infix expression into Reverse Polish Expression.

**public** String infix2Suffix(String expression) {

**if**(expression == **null**) {

**return** **null**;

}

**char**[] chs = expression.toCharArray();

Stack<Character> stack = **new** Stack<Character>();

StringBuilder sb = **new** StringBuilder(chs.length);

**boolean** appendSeparator = **false**;

**boolean** sign = **true**;

**for**(**int** i = 0; i < chs.length; i++) {

**char** c = chs[i];

**if**(c == *BLANK*) {

**continue**;

}

// Next line is used output stack information.

**if**(appendSeparator) {

sb.append(*SEPARATOR*);

appendSeparator = **false**;

}

**if**(isSign(c) && sign) {

sb.append(c);

**continue**;

}

**if**(isNumber(c)) {

sign = **false**;

sb.append(c);

**continue**;

}

**if**(isLeftBracket(c)) {

stack.push(c);

**continue**;

}

**if**(isRightBracket(c)) {

sign = **false**;

**while**(stack.peek() != *LEFT\_BRACKET*) {

sb.append(*SEPARATOR*);

sb.append(stack.pop());

}

stack.pop();

**continue**;

}

appendSeparator = **true**;

**if**(Operator.*isOperator*(c)) {

sign = **true**;

**if**(stack.isEmpty() || stack.peek() == *LEFT\_BRACKET*) {

stack.push(c);

**continue**;

}

**int** precedence = Operator.*getPrority*(c);

**while**(!stack.isEmpty() && Operator.*getPrority*(stack.peek()) >= precedence) {

sb.append(*SEPARATOR*);

sb.append(stack.pop());

}

stack.push(c);

}

}

**while**(!stack.isEmpty()) {

sb.append(*SEPARATOR*);

sb.append(stack.pop());

}

**return** sb.toString();

}

1. **Codes of calculating stack**

This party is mainly used to calculates the result with a stack according to the Reverse Polish Expression.

**public** **double** eval(String expression) {

String str = infix2Suffix(expression);

**if**(str == **null**) {

**throw** **new** IllegalArgumentException("Expression is null!");

}

String[] strs = str.split(" ");

Stack<String> stack = **new** Stack<String>();

/\*Calculate the result of the postfix expression by a stack.\*/

**for**(**int** i = 0; i < strs.length; i++) {

**if**(!Operator.*isOperator*(strs[i])) {

stack.push(strs[i]);

} **else** {

Operator op = Operator.*getInstance*(strs[i]);

**double** right = Double.*parseDouble*(stack.pop());

**double** left = Double.*parseDouble*(stack.pop());

**double** result = op.eval(left, right);

stack.push(String.*valueOf*(result));

}

}

**return** Double.*parseDouble*(stack.pop());

}

# Tests of the calculator

## 5.1 Unit tests

1. **the test of the function of ” Calculator.infix2suffix()”**

Input: 3\*(1+2)^2

Output: 3 1 2 + 2 ^ \*

Input: (3\*(1+2))^2

Output: 3 1 2 + \* 2 ^

Analysis: this test case was used to test the case of blankets and operator priority and proved that the function was right.

1. **the test of the function of ”*Calculator.eval()*”**

Input: 3 1 2 + 2 ^ \*

Output: 27.0

Input: 3 1 2 + \* 2 ^

Output: 81.0

Analysis: this test case proved that the function **”***Calculator.eval()***”** was right.

## 5.2 Integration tests

In this phrase I combine all of the parts of the program, and run the program to test it.

Input: 3\*(1+2)^2

Output: 27.0

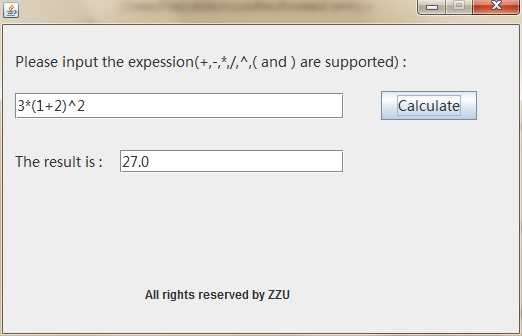
Input: (3\*(1+2))^2

Output: 81.0

Analysis: this test case proved that the program could calculate successfully.

# GUI of the calculator

I just made a simple GUI for the calculator. With the keyboard and mouse the calculator can serve its users successfully.



# Summary

From this program I realized the beauty of Reverse Polish Expression and learned how to use Reverse Polish Expression and calculating stack to develop a calculator.

# Appendix: Development of a Calculator by YACC

## System Environment

1. **Hardware Environment:** A PC , parameters of which are listed as below.

CPU: Intel Core i7 4500U, 1.8GHz.

Memory Capacity: 3GB

1. **Software Environment**: Ubuntu 14.04 (32bit) , Bison.

## Codes of YACC

/\* filename: cal.y \*/

%{

#define YYSTYPE double

#include <math.h>

#include <stdio.h>

#include <ctype.h>

int yylex (void);

void yyerror (char const \*);

%}

%token NUM

%left '-' '+'

%left '\*' '/'

%left NEG

%right '^'

%%

input:

| input line

;

line: '\n'

| exp '\n' { printf ("\t%.10g\n", $1); }

;

exp: NUM { $$ = $1; }

| exp '+' exp { $$ = $1 + $3; }

| exp '-' exp { $$ = $1 - $3; }

| exp '\*' exp { $$ = $1 \* $3; }

| exp '/' exp { $$ = $1 / $3; }

| '-' exp %prec NEG { $$ = -$2; }

| exp '^' exp { $$ = pow ($1, $3); }

| '(' exp ')' { $$ = $2; }

;

%%

int main (void)

{

return yyparse ();

}

int yylex (void)

{

int c;

while ((c = getchar ()) == ' ' || c == '\t');

if (c == '.' || isdigit (c)) {

ungetc (c, stdin);

scanf ("%lf", &yylval);

return NUM;

}

if (c == EOF) return 0;

return c;

}

void yyerror (char const \*s)

{

fprintf (stderr, "%s\n", s);

}

## Compile the codes of YACC

Excute the command below in the console of Ubuntu.

$ yacc cal.y

$ gcc -o cal y.tab.c

Finally a file named “cal”, which was exactly needed , was generated.

## Test the generated file

Excute the command below in the console of Ubuntu.

$ ./cal

Input: 3\*(1+2)^2

Output: 27.0

Input: (3\*(1+2))^2

Output: 81.0

The test proved that developing a calculator by Yacc was successful.