Requirement Analysis Document

Note: all the highlighted part should be submitted by November 10th

Project name

Mathematical Expression Parser and Calculator

Team name

Team 9

Team members

20195061 Kyeong Seop Park

20195073 Junho Park

Scope of the project

The mathematical expression parser that we made provides basic operations such as +, -, *, /, () and some complex operations such as exponential, logarithmic, modular, under root, factorial, and basic trigonometric functions. In addition, regarding the input numbers, we support not only integer and float type, but also special numbers such as PI or e (Euler's constant). The samples of how to use our calculator is in the figure below (Fig. 1). We also handled every possible error and performed the corresponding error recovery. Our calculator parser has three remarkable features. First, our calculator parser gives the answer of the input expression, which makes us easy to check if the precedence rule was correctly applied. Second, we have thought of every possible error and handled all of them; Using the concepts of nondeterministic automata (NFA), we considered all possible error and done error recovery by implementing regular expression in Lex and implementing alternative derivative rules in Yacc. Finally, we stated what error happened in which position of the input.

^	power	3 ^ 2	3^2
v	log	2 v 3	$log_2 3$
m	mod	2 m 3	$2\ mod\ 3 = 2$
r	root	3 r 8	$\sqrt[3]{8}$
e	constant e	e	e
PI	constant pi	PI	π
n	negative sign	n3	-3
a	absolute sign	an3	$\lvert -3 \rvert$
!	factorial	4!	$4!$
t	tangent	t PI	$\tan \pi$
s	sine	s PI	$sin\ \pi$
c	cosine	c PI	$cos\ \pi$

Fig 1. A list of accepted symbols and their examples

Representative diagram to explain our project

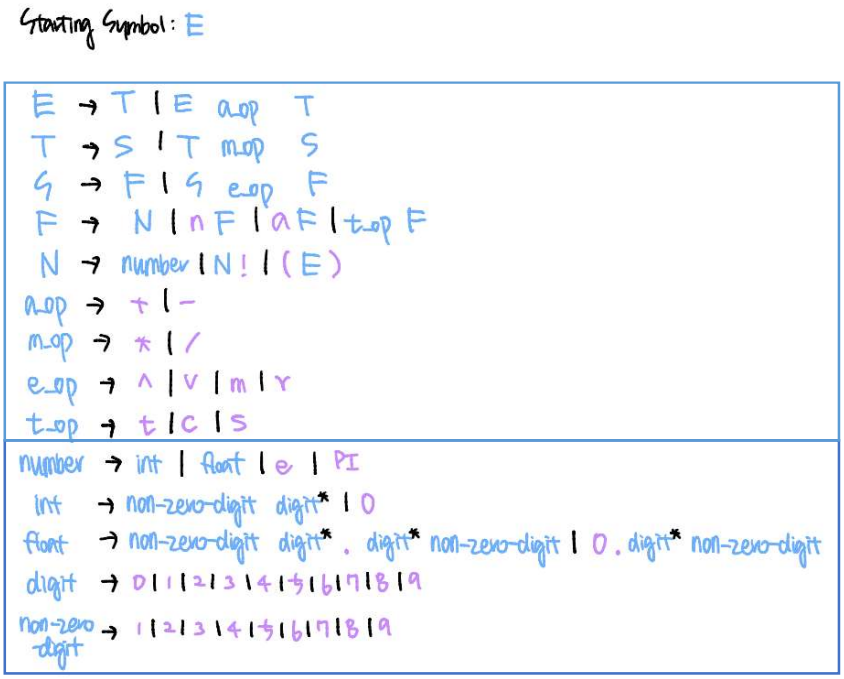


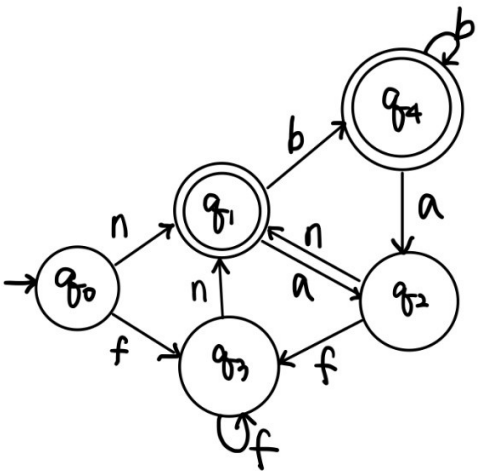
Fig 2. Derivation Rule for Calculator Parser

The derivation rule of our parser is shown on the figure above (Fig. 2). By this rule, precedence is set as () > ! > t_op, a, n > e_op > m_op > a_op. The upper rules are implemented in Yacc, and the below rules are implemented in Lex.

	a_op	m_op	e_op	t_op	number	!	(	)	$	E	T	S	F	N
0				S6	S7		S8			1	2	3	4	5
1	S9								Accept					
2	R1	S11						R1	R1					
3	R2	R2	S13					R2	R2					
4	R3	R3	R3					R3	R3					
5	R4	R4	R4			S18		R4	R4					
6													17	
7	R5	R5	R5			R5		R5	R5					
8				S6	S7		S8			15	2	3	4	5
9				S6	S7		S8				10	3	4	5
10	R6	S11						R6	R6					
11				S6	S7		S8					12	4	5
12	R7	R7	S13					R7	R7					
13				S6	S7		S8						14	5
14	R8	R8	R8					R8	R8					
15	S9							S16						
16	R9	R9	R9			R9		R9	R9					
17	R10	R10	R10					R10	R10					
18	R11	R11	R11			R11		R11	R11					

Fig 3. SLR Parsing Table

The table above is the parsing table using SLR (Fig. 3). Since Lex and Yacc is based on LALR parser, and the expression power of LALR covers SLR, it is enough to show that SLR is satisfied. Since the parsing table has no conflict, we were convinced that the derivation rule was appropriate.



a: [+-*/^vmr]
f: [natcs(]
b: [!)]
n: number

Fig 4. Nondeterministic Automata for Calculator Parser

We have used Lex and Yacc for our implementation. The main concept of our parser is represented on the figure above (Fig. 4). We have divided the states into 4 kinds by what kind of variable has been entered lastly; Q1) Number, Q2) Operator that goes between operands, Q3) Operator that goes in front of an operand, and Q4) Operator that goes in the back of an operand. For any other inputs that can't be accepted by this NFA, they are treated as an error. We organized the types of error, which is represented as the table below (Tab. 1).

State	1	2	3	4	end
0					
1					
2					
3					
4					

Tab 1. Table of Types of Error When Moving State to State

Row State -> Column State
Blue: Accept (Nothing Wrong)
Yellow: Missing Number
Ex) +3-2 , !4^2
Green: Too Much Operators
Ex) ++3 , +!4-2 , n+3!
Red: Missing Operator
Ex) 2 3 , 2! 2+3
White: Blank Expression

Three basic errors are defined; 'Missing Number', 'Too Much Operators', and 'Missing Operator'. For 'Missing Number', we recovered the error by deleting the operator if the number was missing in front of the operator, or put 1 if the number was missing after the operator. For 'Too Much Symbol', we deleted

all operators except the first operator. For ‘Missing Symbol’, we put * operator between operands. Parentheses were treated specially, since they are slightly different from other operators. Thus, we handled associated errors; ‘Missing Opening Parenthesis’, ‘Missing Closing Parenthesis’, and ‘Useless Parentheses’. For ‘Missing Opening Parenthesis’, we deleted the closing parenthesis, and for ‘Missing Closing Parenthesis’, we added closing parenthesis. For ‘Useless Parentheses’, we replaced ‘()’ with 1. For unexpected symbols, we treated them as error ‘unexpected Character’, and changed them to 1. We have also represented some ‘Semantic Warning’, and ‘Semantic Error’.

Limitations of our project

Floating point exceptions couldn’t be handled because of the fundamental floating-point arithmetic (Fig. 5). Also, our calculator can only handle these operations; basic (+,-,*,/), exponential, logarithmic, modular, under root, factorial, and basic trigonometric operations. Some functions such as inverse trigonometric functions are not implemented in our calculator. Since the type of the number is set to double, the size of the number is limited to the range of double. If it exceeds the range, it returns INF.

```
========Enter Expression : s PI
========Answer is 1.22465e-16
```

Fig 5. Output of $sin\pi$

Sample input and output

Any kind of mathematical expressions can be the input, so we demonstrate one specific correct input, and incorrect inputs for each error to show how error handling is done.

```
pjh001024@DESKTOP-TKNTB23:~/Compiler$ ./cal
This is TEAM 9 Calculator by 20195061 KyeongSeop Park, 20195073 Junho Park.
Enter your expression and press 'Enter'.
Available number: integer, real number, e, PI
Available operator: +, -, *, /, ^, v, m, r, n, a, t, c, s, !, (, )
Enter "quit" to quit the calculator

========Enter Expression : 2^(an3!+3)+5*s(PI/2)
========Answer is 517
```

$$2^{(|-(3!)|+3)} + 5 * sin\frac{\pi}{2} = 2^9 + 5 = 517$$

Fig 6. Example of Accepted Expression

1) Missing Number

```
========Enter Expression : +2
ERROR!! MISSING NUMBER
Recovered by deleting operator '+'
========Answer is 2
```

2) Too Much Operators

```
========Enter Expression : 3+*+3
ERROR!! TOO MUCH OPERATORS: '+*+'
Recovered by deleting operators '+*+' except '+'
========Answer is 6
```

3) Missing Operator

```
========Enter Expression : 3 3+4
ERROR!! MISSING OPERATOR BETWEEN 3.000000 AND 3.000000
Recovered by putting '*' between 3.000000 and 3.000000
========Answer is 13
```

4) Useless Parentheses

```
========Enter Expression : ()+4
ERROR!! USELESS PARENTHESES '()'
Recovered by replacing '()' with 1
========Answer is 5
```

5) Missing Opening Parenthesis

```
========Enter Expression : 3+3)*2)
ERROR!! MISSING OPENING PARENTHESIS '('
Recovered by deleting closing parenthesis ')'
ERROR!! MISSING OPENING PARENTHESIS '('
Recovered by deleting closing parenthesis ')'
========Answer is 9
```

6) Missing Closing Parenthesis

```
========Enter Expression : ((3+4)+(3
ERROR!! MISSING 2 CLOSING PARENTHESIS ')'
Recovered by putting 2 closing parenthesis ')'
========Answer is 10
```

7) Unexpected Character

```
========Enter Expression : #+1
ERROR!! UNEXPECTED CHARACTER : '#'
Recovered by replacing '#' with 1
========Answer is 2
```

8) Semantic Warning

```
========Enter Expression : 2 / 0
Semantic WARNING: DIVIDED BY ZERO
========Answer is inf
```

9) Semantic Error

```
========Enter Expression : (n3)!
Semantic ERROR!! NEGATIVE FACTORIAL
Recovered by replacing -3.000000 with |-3.000000|
========Answer is 6
```

+ Displaying Multiple Errors

```
========Enter Expression : (3-2**9)+3*(2
ERROR!! TOO MUCH OPERATORS: '**'
Recovered by deleting operators '**' except '*'
ERROR!! MISSING 1 CLOSING PARENTHESIS ')'
Recovered by putting 1 closing parenthesis ')'
========Answer is -9
```

Fig 7. Examples of Error Recovery

Distribution of our work

We have equal contribution to our project, since we had met and done every task together, such as designing and implementing derivation rule, thinking about the concept of error handling, implementing error detecting and recovery, and making presentation materials (PowerPoint Slide / Requirement Analysis Document).