REPORT

<Compiler Assignment>

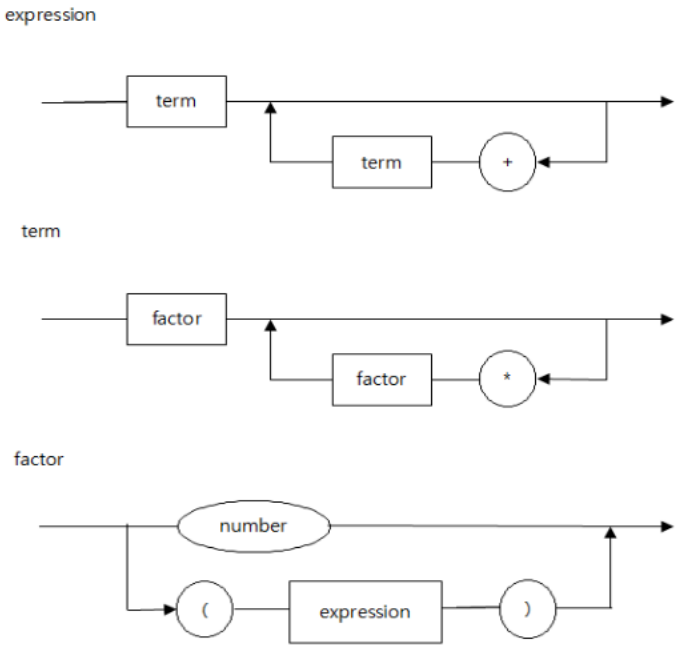
과제 1 : 수식의 값 계산

과 목	컴파일러
담당 교수	유재우 교수님
제출 일자	2024 년 3 월 11 일
전공	소프트웨어학부
학번	20182745
이름	정호재

1. 과제 내용

1차 과제로 Recursive Descent Parsing 방식을 이용하여 수식의 값을 계산하는 프로그램을 작성하였다. 사용자가 수식을 입력하면 프로그램은 수식을 token화 하여 실수를 계산한다.과제 내용에 더해 뺄셈과 나눗셈 연산이 가능하도록 기능을 추가하였다.

2. 해결방법



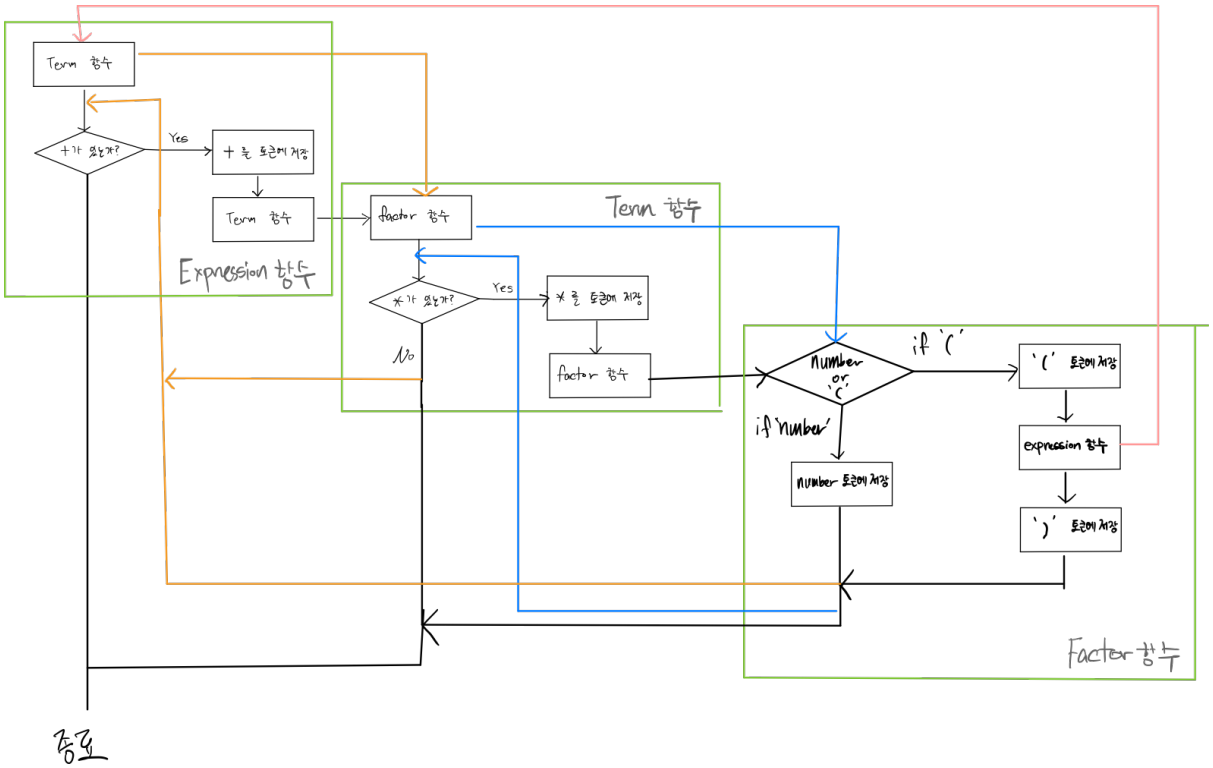
위 사진에 총 3개의 syntax graph가 있다. 본 과제는 사진에 첨부된 3개의 graph로 수식의 값을 계산한다. Parsing 방식의 이름에서 알 수 있듯이 재귀 하강 방식으로 수식을 분석한다. 위 그림을 바탕으로 설명하면 다음과 같다.

- Expression 함수 호출
    - Term 함수 호출
        - Factor 함수 호출
            - 정수나 실수가 있으면 Token에 저장
            - 정수나 실수가 없으면 '('를 Token에 저장하고 Expression 호출
            - Expression이 마치면 ')'를 Token에 저장하고 Factor return 값 반환

- '*' (또는 '/') 가 있다면 Factor 함수를 호출

- '+' (또는 '-') 가 있다면 Term 함수를 호출

위 과정을 순서도로 나타내면 다음과 같다.



정호

3. 결론

컴파일러의 논리 구조와 비슷하면서도 사칙연산이 가능한 간단한 프로그램을 완성하였다.

4. 프로그램 실행 결과 (스크린 샷)

```
hojae@dl-server:~/Desktop/SSU_5-1/compiler/1st_assignment$ ./first_assignment
Enter an arithmetic expression: 3 + 4.2 * 1.8 + 4 * (2 + 3.2)
31.359999
```

5. Source Code

<Github link> : https://github.com/JEONG-HO-JAE/Introduction_of_Compiler/blob/main/1st_assignment/first_assignment.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
typedef enum {NULL_VAL, INTEGER, FLOAT} ValueType;
typedef enum {NUMBER, PLUS, MINUS, STAR, DIVISION, LP, RP, END} TokenType;

typedef struct {
    TokenType tokenType;
    ValueType valueType;
    union {
        int intValue;
        float floatValue;
    } Value;
} Token;

Token token;
char curChar;

Token expression();
Token term();
Token factor();
void get_token();
void error(int);

Token expression() {
    Token token_from_term_1 = term();

    while (token.tokenType == PLUS || token.tokenType == MINUS) {
        TokenType op = token.tokenType;
        get_token();
        Token token_from_term_2 = term();

        if (op == PLUS) {
            if(token_from_term_1.valueType == INTEGER &&
token_from_term_2.valueType == INTEGER) {
                token_from_term_1.Value.intValue = token_from_term_1.Value.intValue
+ token_from_term_2.Value.intValue;
                token_from_term_1.valueType = INTEGER;
            }
            else if (token_from_term_1.valueType == INTEGER &&
token_from_term_2.valueType == FLOAT) {
                token_from_term_1.Value.floatValue =
token_from_term_1.Value.intValue + token_from_term_2.Value.floatValue;
                token_from_term_1.valueType = FLOAT;
            }
            else if (token_from_term_1.valueType == FLOAT &&
token_from_term_2.valueType == INTEGER) {
                token_from_term_1.Value.floatValue =
token_from_term_1.Value.floatValue + token_from_term_2.Value.intValue;
```

```
                token_from_term_1.valueType = FLOAT;
            }
            else {
                token_from_term_1.Value.floatValue =
token_from_term_1.Value.floatValue + token_from_term_2.Value.floatValue;
                token_from_term_1.valueType = FLOAT;
            }
        } else {
            if(token_from_term_1.valueType == INTEGER &&
token_from_term_2.valueType == INTEGER) {
                token_from_term_1.Value.intValue = token_from_term_1.Value.intValue
- token_from_term_2.Value.intValue;
                token_from_term_1.valueType = INTEGER;
            }
            else if (token_from_term_1.valueType == INTEGER &&
token_from_term_2.valueType == FLOAT) {
                token_from_term_1.Value.floatValue =
token_from_term_1.Value.intValue - token_from_term_2.Value.floatValue;
                token_from_term_1.valueType = FLOAT;
            }
            else if (token_from_term_1.valueType == FLOAT &&
token_from_term_2.valueType == INTEGER) {
                token_from_term_1.Value.floatValue =
token_from_term_1.Value.floatValue - token_from_term_2.Value.intValue;
                token_from_term_1.valueType = FLOAT;
            }
            else {
                token_from_term_1.Value.floatValue =
token_from_term_1.Value.floatValue - token_from_term_2.Value.floatValue;
                token_from_term_1.valueType = FLOAT;
            }
        }
    }
    return token_from_term_1;
}

Token term() {
    Token token_from_factor_1 = factor();
    while (token.tokenType == STAR || token.tokenType == DIVISION) {
        TokenType op = token.tokenType;
        get_token();
        Token token_from_factor_2 = factor();
        if (op == STAR) {
            if (token_from_factor_1.valueType == INTEGER &&
token_from_factor_2.valueType == INTEGER) {
            token_from_factor_1.Value.intValue = token_from_factor_1.Value.intValue
* token_from_factor_2.Value.intValue;
```

```
            token_from_factor_1.valueType = INTEGER;
            }
            else if (token_from_factor_1.valueType == INTEGER &&
token_from_factor_2.valueType == FLOAT) {
                token_from_factor_1.Value.floatValue =
token_from_factor_1.Value.intValue * token_from_factor_2.Value.floatValue;
                token_from_factor_1.valueType = FLOAT;
            }
            else if (token_from_factor_1.valueType == FLOAT &&
token_from_factor_2.valueType == INTEGER) {
                token_from_factor_1.Value.floatValue =
token_from_factor_1.Value.floatValue * token_from_factor_2.Value.intValue;
                token_from_factor_2.valueType = FLOAT;
            }
            else {
                token_from_factor_1.Value.floatValue =
token_from_factor_1.Value.floatValue * token_from_factor_2.Value.floatValue;
                token_from_factor_2.valueType = FLOAT;
            }
        } else {
            if (token_from_factor_2.valueType == 0) {
                error(5);
            }
            if(token_from_factor_1.valueType == INTEGER &&
token_from_factor_2.valueType == INTEGER) {
                token_from_factor_1.Value.intValue =
token_from_factor_1.Value.intValue / token_from_factor_2.Value.intValue;
                token_from_factor_1.valueType = INTEGER;
            }
            else if (token_from_factor_1.valueType == INTEGER &&
token_from_factor_2.valueType == FLOAT) {
                token_from_factor_1.Value.floatValue =
token_from_factor_1.Value.intValue / token_from_factor_2.Value.floatValue;
                token_from_factor_1.valueType = FLOAT;
            }
            else if (token_from_factor_1.valueType == FLOAT &&
token_from_factor_2.valueType == INTEGER) {
                token_from_factor_1.Value.floatValue =
token_from_factor_1.Value.floatValue / token_from_factor_2.Value.intValue;
                token_from_factor_1.valueType = FLOAT;
            }
            else {
                token_from_factor_1.Value.floatValue =
token_from_factor_1.Value.floatValue / token_from_factor_2.Value.floatValue;
                token_from_factor_2.valueType = FLOAT;
            }
        }
```

```
    }
    return token_from_factor_1;
}

Token factor() {
    Token result;
    if (token.tokenType == NUMBER) {
        result = token;
        // if (token.valueType==INTEGER)
        //     printf("%d", result.Value.intValue);
        // else printf("%lf", result.Value.floatValue);
        get_token();
        if (token.tokenType == LP) error(4);
    }
    else if (token.tokenType == LP) {
        get_token();
        result = expression();
        if (token.tokenType == RP) get_token();
        else error(2);
    }else error(1);
    return result;
}

void get_token() {
    int intValue = 0;
    float floatValue = 0.0;

    while ((curChar == ' ') || (curChar == '\t')) curChar = getchar();
    if (curChar == '+') { curChar = getchar(); token.tokenType = PLUS;
token.valueType = NULL_VAL; return; }
    if (curChar == '-') { curChar = getchar(); token.tokenType = MINUS;
token.valueType = NULL_VAL; return; }
    if (curChar == '*') { curChar = getchar(); token.tokenType = STAR;
token.valueType = NULL_VAL; return; }
    if (curChar == '/') { curChar = getchar(); token.tokenType = DIVISION;
token.valueType = NULL_VAL; return; }
    if (curChar == '(') { curChar = getchar(); token.tokenType = LP;
token.valueType = NULL_VAL; return; }
    if (curChar == ')') { curChar = getchar(); token.tokenType = RP;
token.valueType = NULL_VAL; return; }
    if (curChar == '\n') { token.tokenType = END; token.valueType = NULL_VAL;
return; }

    if ((curChar >= '0') && (curChar <= '9')) {
        token.tokenType = NUMBER;
        token.valueType = INTEGER;
        do {
```

```
            intValue = intValue * 10 + curChar - '0';
            curChar = getchar();
        } while ((curChar >= '0') && (curChar <= '9'));
        token.Value.intValue = intValue;

        if (curChar == '.') {
            token.valueType = FLOAT;
            curChar = getchar(); // jump '.' char
            float fraction = 0.1;

            while ((curChar >= '0') && (curChar <= '9')) {
                floatValue += (curChar - '0') * fraction;
                fraction *= 0.1;
                curChar = getchar();
            }
            token.Value.floatValue = intValue + floatValue;
        } else {
            token.Value.intValue = intValue;
        }
    }
}

void error(int i){
    switch (i) {
        case 1: printf("Syntax Error: Missing '('\n"); break;
        case 2: printf("Syntax Error: Missing ')'\n"); break;
        case 3: printf("Syntax Error: Unexpected End of Expression\n"); break;
        case 4: printf("Syntax Error: Missing '+' or '*'\n"); break;
        case 5: printf("Error: Division by zero\n"); break;
    }
    exit(1);
}

int main(void){
    Token result;
    curChar = ' ';
    printf("Enter an arithmetic expression: ");
    get_token();
    result = expression();
    // if (result.valueType==INTEGER)
    //         printf("%d", result.Value.intValue);
    // else printf("%lf", result.Value.floatValue);
    if (token.tokenType != END) error(3);
    else result.valueType==INTEGER?printf("%d\n", result.Value.intValue) :
printf("%lf\n", result.Value.floatValue);
    return 0;
}
```