1. 코드

```
#include <stdio.h>

enum {NULLPTR, NUMBER, PLUS, STAR, LP, RP, END} token;

float num = 0.0;
int flag_int = 1, flag_float = 0;
int position = 0;
int position_warning;
int prev_token;

float expression();
float term();
float factor();
void get_token();
void error(int i);

void main() {
   float result;
   get_token();
   result = expression();

   //if(token != END) error(3);
   //else {
       if(flag_int && flag_float) error(4);

       if(!flag_float) printf("%d \n", (int)result);
       else printf("%f \n", result);
   //}
}

float term() {
   float result;
   result = factor();

   while(token == STAR) {
       get_token();
       result = result * factor();
   }
```

```
    return result;
}

float expression() {
    float result;
    result = term();

    while(token == PLUS) {
        get_token();
        result = result + term();
    }

    return result;
}

float factor() {
    float result;

    if(token == NUMBER) {
        result = num;
        get_token();
        num = 0;
    }
    else if (token == LP) {
        get_token();
        result = expression();

        if(token == RP) get_token();
        else error(2);
    }
    else error(1);

    return result;
}

void get_token() {
    /* TODO */
    // next_token => token
    // number value => num
```

```
    char next_token;
    float op = -1.0;

    if(token == NUMBER) next_token = prev_token;
    else next_token = getchar();
    position++;

    if('0' <= next_token && next_token <= '9') {
        token = NUMBER;

        while(('0' <= next_token && next_token <= '9') || next_token ==
'.') {
            if(next_token == '.') {
                op = 1;
                flag_float = 1;
                position_warning = position;
            }
            else {
                if(op == -1.0) num = num*10 + (next_token - '0'); /*
integer */
                else {  /* float */
                    op /= 10;
                    num += (next_token - '0') * op;
                }
            }

            next_token = getchar();
        }
        prev_token = next_token;
    } /* make num */

    else {
        switch(next_token) {
        case '+':
            token = PLUS;
            break;
        case '*':
            token = STAR;
            break;
        case '(':
```

```
            token = LP;
            break;
        case ')':
            token = RP;
            break;
        case EOF:
            token = END;
            break;
        default:
            token = NULLPTR;
            break;
        }
    }

    return;
}

void error(int i) {
    switch(i) {
    case 1:
        printf("factor는 숫자거나 (로 시작해야 합니다\n");
        break;
    case 2:
        printf("factor는 )로 끝나야 합니다\n");
        break;
    case 3:
        printf("수식이 끝나지 않았습니다\n");
        break;
    case 4:
        printf("WARNING: 위치 %d 부근에서 피연산자 형식 혼합됨\n",
position_warning);
    }
}
```

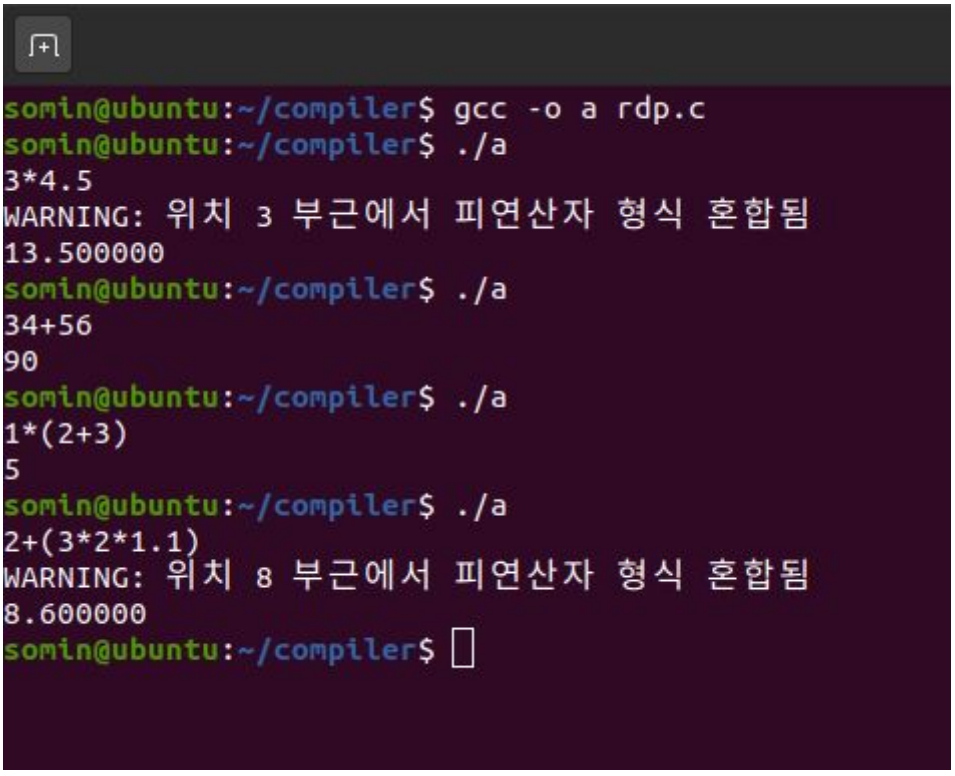
2. 문제 해결 방법

get_token()에서 +, *, (,)를 받으면 token에 알맞은 값을 할당해서 expression(), term(), factor()에서 사용할 수 있게 했습니다.

가장 고민했던 부분은 숫자가 들어왔을 때 처리하는 부분이였는데, 숫자를 사용하는 factor()에서는 token이 NUMBER일 때 num을 한 번만 읽어오므로 get_token()에서 숫자가 끝날 때까지 여러 번 받아 num을 완성해야 한다고 생각했습니다. get_token()에서 읽었는데 숫자라면 while문 안에서 숫자 혹은 소수점이 아닌 문자를 받을 때까지 계속 문자를 받습니다.

소수점이 안 나왔으면 정수 부분이므로 전 num값에 10을 곱하고 새로운 값을 더해가며 num을 완성했고, 소수점이 나오면 새로운 값에 0.1, 0.01, ...씩 곱해서 소수 부분을 완성합니다.
계속 숫자를 받다가 0~9 혹은 소수점이 아닌 문자가 들어오면 num을 완성했으므로 get_token()을 끝내야 하므로 while문을 탈출합니다. 숫자가 끝난 이후 get_token()을 실행하면 while문을 나오기 직전 받은 문자를 무시하고 새로 받게 되는 것이기 때문에 이 경우 미리 전역변수에 저장해뒀다가 다시 불러왔습니다.
result값은 다 float 형식이어서 만약 실수가 들어오지 않았다면 정수 형태로 출력하고, 실수가 들어왔다면 실수 형태로 출력했습니다.
그리고 실수가 입력됐다면 소수점이 들어오는 마지막 위치를 기억해뒀다가 어느 위치에서 피연산자가 혼합되는지 warning을 띄워줍니다.

3. 수행 결과



위의 사진에서 실수랑 정수가 혼합된 수식, 혼합되지 않은 수식, 괄호가 있는 수식이 잘 계산되고 있습니다.