2022056262_Project1

과제 목표

Lex, C code로 C-minus scanner 만들기

컴파일 환경 및 방법

- Ubuntu 20.04.4 LTS
- 첨부되어 있던 Makefile을 사용 gcc로 컴파일, Makefile로 빌드

작동 방식

main.c를 분석해본 결과, command line 인자로 input file을 넣으면 각종 검사를 진행한 후에 (scanner 구성과 무관) while문으로 getToken이 ENDFILE을 return할 때까지 getToken을 실행한다.

`while (getToken()!=ENDFILE);`
(우리의 경우 PARSE를 FALSE로 설정했기 때문에 if NO_PARSE에 걸리는 것) 해당 함수는 scan.c, cminus.l 파일에 각각 존재한다. getToken 실행 방식은 추후 method1, 2 설명에 작성할 것이다.
ENDFILE을 return하면 input file을 닫고 종료한다.

Method 1: C code

states

```
typedef enum
   { START,INEQ,INCOMMENT,INNE,
   INLT,INGT,INNUM,INID,DONE,
   INCOMMENT_ }
   StateType;
```

input 하나를 받았을 때 token을 정하기 모호한 경우에 transition할 state들을 추가

INOVER은 /을 받았을 때 바로 확인해서 OVER 혹은 INCOMMENT로 가도록 작성하여서 추가하지 않음

getToken 함수 초반부

```
TokenType getToken(void)
{  int tokenStringIndex = 0;
   TokenType currentToken;
   StateType state = START;
      int save;
   while (state != DONE)
   { int c = getNextChar();
    int next;
     save = TRUE;
```

초기 index를 0, 초기 state를 START로 설정하고 반환할 tokentype 변수 선언, save로 저장 여부 선택. getToken 함수를 보면 save가 참이고 index가 최대를 벗어나지 않았을 때만 tokenString을 저장한다.
state가 DONE 즉, token 하나가 완전히 받아들여지지 않은 상태에선 다음 문자열을 받고 기록을 위해 save를 참으로 설정한다.

getToken 함수 state가 START인 경우

```
//state가 START인 part 1
switch (state)
     {  case START:
         if (isdigit(c))
           state = INNUM;
         else if (isalpha(c))
           state = INID;
         else if (c=='!')
          state = INNE;
```

길이상 중요한 부분만 발췌한 코드이다. state마다 처리하는 방식이 다른데, START 즉 처음 input을 받은 상태에선 숫자, 문자, 그 외 특수문자들에 따라서 다른 행동을 취한다.

숫자라면 우선 임시 숫자의 의미로 INNUM state로 설정한다. 문자 역시 마찬가지이다.
!의 경우 느낌표 혹은 NE가 될 수 있기 때문에 이 역시 INNE로 임시 상태로 지정해준다.
!와 동일한 경우가 >,<,=등이 있다. 모두 INNE처럼 초기 선언한 state에 대응시켜주면 된다.

```
//state가 START인 part 2
//incomment_는 incomment 상태에서만
//가능하기 때문에 이 경우엔 바로 곱셈
 else if (c == '*') {
    state = DONE;
    currentToken = TIMES;
    }
else if((c=='')||(c=='t')||(c=='\n')){
    save = FALSE;
/* /를 받으면 over일 수도 주석일 수도
/있으니 case 나눠서 state 저장*/
     else if (c == '/')
       {
       next=getNextChar();
        if(next=='*'){
/*기존 tiny에서 처리하던 문법과 동일*/
           save = FALSE;
            state = INCOMMENT;
          }
          else{
```

```
            ungetNextChar();
             state = DONE;
            currentToken = OVER;
          }
        }
```

comment를 처리하는 방식이다.

*는 곱셈 혹은 */로 닫는 주석이 될 수 있다. 그러나 여기선 state가 START이므로 곱셈만 가능하기 때문에 state를 DONE으로, 현재 토근을 곱셈으로 지정해준다.

공백, tab 등은 무시하기 때문에 그냥 저장하지 않고 넘기면 된다.

/인 경우 나눗셈 혹은 주석이 가능하기 때문에 getNextChar 함수로 다음 문자를 확인한다. *라면 주석이므로 INCOMMENT 상태와 함께 주석은 저장할 필요가 없으므로 save를 거짓으로 바꿔준다. 그렇지 않다면 단순 나눗셈이므로 state를 DONE으로, 현재 토큰을 OVER로 바꾸고 방금 받은 문자는 다시 분석해야 하기 때문에 ungetNextChar 함수로 다시 돌려준다.

```
//state가 START인 part 3
else{
 /*TOKEN 결정 나니까 DONE으로 수정*/
      state = DONE;
      switch (c)
      { case EOF:
          save = FALSE;
           currentToken = ENDFILE;
              break;

            case '+':
              currentToken = PLUS;
              break;
```

첫 input만으로 token이 정확해지는 경우는
state를 DONE으로 바꾸고 token도 바로 지정해서 끝내주면 된다.

COMMENT 관련 state

```
/* 주석 내부 상태 / *를 받은 상태! */
       case INCOMMENT:
         save = FALSE;
         if (c == EOF)
         { state=DONE;
               currentToken = ENDFILE;
         }
         else if (c == '*'){
          state = INCOMMENT_;
         }
         break;
       case INCOMMENT_:
        save = FALSE;
        if (c == '/'){ state = START; }
        else if (c == EOF) {
        state = DONE;
        currentToken = ENDFILE;
    }
        else if (c=='*') {state =INCOMMENT_;}
        else {state = INCOMMENT;}
        break;
```

주석이라 저장할 필요가 없으니 save를 false로 지정한다. 주석이 열린 상태에서 EOF가 들어오면 state를 DONE으로 바꾸고 currentToken을 ENDFILE로 지정해주면 된다. 만약 *를 받으면 닫는 주석일 수도 있기 때문에 상태를 INCOMMENT_로 변경한다. INCOMMENT_에서는 /를 받는 경우, *를 받는 경우, 그 외의 경우로 나눌 수 있는데 /를 받으면 주석이 닫힌 거라서 state를 다시 START로 만들어준다. while문을 다시 돌 때 save가 TRUE로 바뀌기 때문에 여기서 save를 변경할 필요는 없다.

*를 다시 받는 경우는 여전히 INCOMMENT_ 상태기 때문에 state를 유지해주면 된다.

그 외의 경우는 아직 주석이라는 의미기 때문에 state를 INCOMMENT로 변경한다. EOF를 받는다면

그 외 state

```
case INNE:
         state = DONE;
         if (c == '=')
           currentToken = NE;
          else if (c == EOF) {
            state = DONE;
            currentToken = ENDFILE;
          }
         else
         { /* backup in the input */
           ungetNextChar();
           save = FALSE;
           currentToken = ERROR;
         }
         break;
```

cminus에선 !를 단독 사용하지 않기 때문에 다음에 받은 게 =면 정상 처리, 그 외는 ERROR를 반환한다.

```
case INLT:
         state = DONE;
         if (c == '=')
           currentToken = LE;
          else if (c == EOF) {
            state = DONE;
            currentToken = ENDFILE;
          }
         else
         { /* backup in the input */
           ungetNextChar();
           save = FALSE;
           currentToken = LT;
         }
         break;
```

<,>,= 등은 이와 같은 방식으로 올바른 token을 지정해주면 된다.

```
case INID:
         if (isalpha(c)||isdigit(c)){
          state = INID;
         }
         else if (c == EOF) {
            state = DONE;
            currentToken = ENDFILE;
          }
         else
         { /* backup in the input */
           ungetNextChar();
           save = FALSE;
           state = DONE;
           currentToken = ID;
         }
         break;
```

기존 코드는 식별자에 숫자가 포함되는 경우를 배제해서 수정해주었다.

token 저장

```
//save가 참이고 현재 index가 최대를 벗어나지 않는다면
//tokenString에 방금 읽은 문자 저장
if ((save) && (tokenStringIndex <= MAXTOKENLEN))
   tokenString[tokenStringIndex++] = (char) c;
//만약 token이 완성됐다면 null 추가해서 c 문자열로 만들기
if (state == DONE){
 tokenString[tokenStringIndex] = '\0';
 //그리고 만약 token이 ID라면 예약어인지 확인 과정 거침
 //reservedLookup의 자료형은 tokenType이므로 currentToken이
 //ID거나 예약어 그 자체가 됨
 if (currentToken == ID)
    currentToken = reservedLookup(tokenString);
 }
 //여기서 printToken 호출해서 출력하고 token을 return한다.
```

Method 2: Lex(Flex)

definition

```
digit       [0-9]
number      {digit}+
letter      [a-zA-Z]
identifier {letter}({letter}|{digit})*
newline     \n
whitespace  [ \t]+
```

기존 tiny 파일에서 identifier만 수정해주었다. 기존 코드에선 식별자에 숫자가 포함이 안 되었기 때문에 숫자도 포함해서 인식하도록 수정

rule

```
"int"              { return INT; }
"if"               { return IF; }
"else"             { return ELSE; }
"/"                { return OVER; }
"<"                { return LT; }
.
.
"{"              { return LCURLY; }
"}"              { return RCURLY; }
{number}           { return NUM; }
{identifier}       { return ID; }
{newline}          { lineno++; }
{whitespace}{/* skip whitespace */}
.                  {return ERROR; }
<<EOF>>          {return ENDFILE; }
```

ppt에 주어진 예약어들을 추가하였다.

주석 처리 rule

```
"/*" {
    char current;
    char next;
    while (1) {
      current = input();
      /* 만나면 다음 input /이거인지 확인 */
      if (current == '*') {
        next=input();
        if(next=='/') break;
        else unput(next);
      }
      // 주석 다 안 끝나고 EOF 만나면 그냥 정상 종료...
      else if (current == 0 || current == EOF) {
        return ENDFILE;
      }

      else if (current == '\n') {
        lineno++;
      }
    }
}
```

주석이 열렸을 때 *를 만나면 다음 input을 확인하고, /라면 break로 끝내면 되고 그게 아니라면 input을 빼고 다시 while문을 돌면 된다. 만약 EOF를 만난다면 error가 아니라 정상 종료를 시키면 된다. 이렇게 작성하면 /* /* */ */ 이런 경우에 문제 없이 *과 /를 tokenizing할 수 있다.

/* * */* 이런 경우에도 정상적으로 주석 종료 후 *를 tokenizing할 수 있다.

getToken 실행 방식

```
TokenType getToken(void)
{ static int firstTime = TRUE;
  TokenType currentToken;
  if (firstTime)
  { firstTime = FALSE;
    lineno++;
    yyin = source;
    yyout = listing;
  }
  currentToken = yylex();
  strncpy(tokenString,yytext,MAXTOKENLEN);
  if (TraceScan) {
    fprintf(listing,"\t%d: ",lineno);
    printToken(currentToken,tokenString);
  }
```

```
  return currentToken;
}
```

처음 시작하면 lineno을 1 증가하고 input file과 output file을 지정한다. yyin, yyout, yytext등의 변수는 lex.yy.c file에서 확인할 수 있다. (cscope 사용) yylex 함수 자체의 정의는 찾지 못 했으나 조사를 통해 flex가 자동으로 만들어주는 state machine임을 알 수 있었다. 즉 yylex의 return값은 현재 token의 type이다. strncpy로 현재 token을 tokenString에 저장해주고 printToken 함수를 통해 output을 출력한다. (TraceScan은 현재 scanner를 만들고 있으므로 true로 사전 설정을 해준 상태이다.)

Test Case

comment 관련 testCase (주석 완료 전 EOF, 주석 사이에 /나 *가 들어간 경우, 중첩 주석)

```
/*
```

```
juri@juri-VirtualBox:~/COMPILERS2024/project1$ ./cminus_lex testComment1.txt

TINY COMPILATION: testComment1.txt
        2: EOF
```

```
/* hello * bye~ */
```

```
juri@juri-VirtualBox:~/COMPILERS2024/project1$ ./cminus_lex testComment2.txt

TINY COMPILATION: testComment2.txt
        2: EOF
```

```
/*hello/*world*/ */
```

```
juri@juri-VirtualBox:~/COMPILERS2024/project1$ ./cminus_lex testComment3.txt

TINY COMPILATION: testComment3.txt
        1: *
        1: /
        2: EOF
```

```
juri@juri-VirtualBox:~/COMPILERS2024/project1$ ./cminus_cimpl testComment3.txt

TINY COMPILATION: testComment3.txt
        1: *
        1: /
        2: EOF
```

id에 숫자가 포함된 testCase

```
void main(void){
int i34e; int x[5];
    i34e = 0;
    return;}
```

```
juri@juri-VirtualBox:~/COMPILERS2024$ ./cminus_lex testID.txt

TINY COMPILATION: testID.txt
        1: reserved word: void
        1: ID, name= main
        1: (
        1: reserved word: void
        1: )
        2: {
        3: reserved word: int
        3: ID, name= i34e
        3: ;
        3: reserved word: int
        3: ID, name= x
        3: [
        3: NUM, val= 5
        3: ]
        3: ;
        5: ID, name= i34e
        5: =
        5: NUM, val= 0
        5: ;
        6: reserved word: return
        6: ;
        7: }
        8: EOF
```

```
juri@juri-VirtualBox:~/COMPILERS2024$ ./cminus_cimpl testID.txt

TINY COMPILATION: testID.txt
        1: reserved word: void
        1: ID, name= main
        1: (
        1: reserved word: void
        1: )
        2: {
        3: reserved word: int
        3: ID, name= i34e
        3: ;
        3: reserved word: int
        3: ID, name= x
        3: [
        3: NUM, val= 5
        3: ]
        3: ;
        5: ID, name= i34e
        5: =
        5: NUM, val= 0
        5: ;
        6: reserved word: return
        6: ;
        7: }
        8: EOF
```

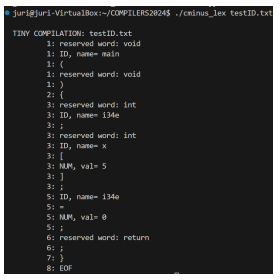
기본 제공 testCase

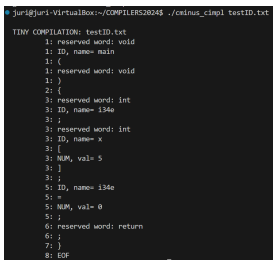
```
juri@juri-VirtualBox:~/COMPILERS2024$ ./cminus_lex  test.1.txt

TINY COMPILATION: test.1.txt
        4: reserved word: int
        4: ID, name= gcd
        4: (
        4: reserved word: int
        4: ID, name= u
        4: ,
        4: reserved word: int
        4: ID, name= v
        4: )
        5: {
        6: reserved word: if
        6: (
        6: ID, name= v
        6: ==
        6: NUM, val= 0
        6: )
        6: reserved word: return
        6: ID, name= u
        6: ;
7: reserved word: else
7: reserved word: return
7: ID, name= gcd
7: (
7: ID, name= v
7: ,
7: ID, name= u
7: -
7: ID, name= u
7: /
7: ID, name= v
7: *
7: ID, name= v
7: )
7: ;
9: }
```

```
11: (
11: reserved word: void
11: )
12: {
13: reserved word: int
13: ID, name= x
13: ;
13: reserved word: int
13: ID, name= y
13: ;
14: ID, name= x
14: =
14: ID, name= input
14: (
14: )
14: ;
```

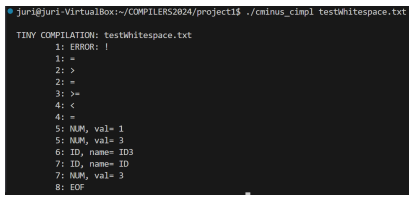
```
14: ID, name= y
14: =
14: ID, name= input
14: (
14: )
14: ;
15: ID, name= output
15: (
15: ID, name= gcd
15: (
15: ID, name= x
15: ,
15: ID, name= y
15: )
15: )
15: ;
16: }
17: EOF
```





whitespace testCase

```
!                    =    /* error, =*/
>                =        /* >,= */
>=                        /* >= */
<       =                 /* <, = */
1 3                       /* num, num */
ID3                       /* id */
ID 3                      /* id, num */
```

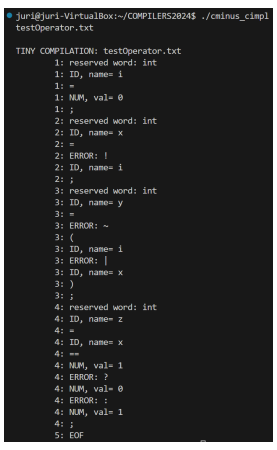


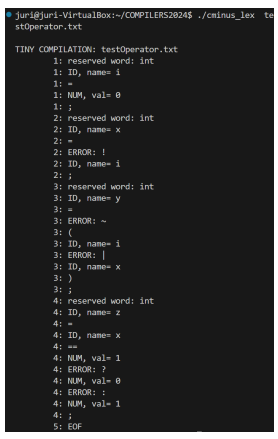

```
juri@juri-VirtualBox:~/COMPILERS2024/project1$ ./cminus_lex testWhitespace.txt

TINY COMPILATION: testWhitespace.txt
        1: ERROR: !
        1: =
        2: >
        2: =
        3: >=
        4: <
        4: =
        5: NUM, val= 1
        5: NUM, val= 3
        6: ID, name= ID3
        7: ID, name= ID
        7: NUM, val= 3
        8: EOF
```

잘못된 token testCase

```
int i=0;
int x=!i;
int y=~(i|x);
int z=x==1?0:1;
```

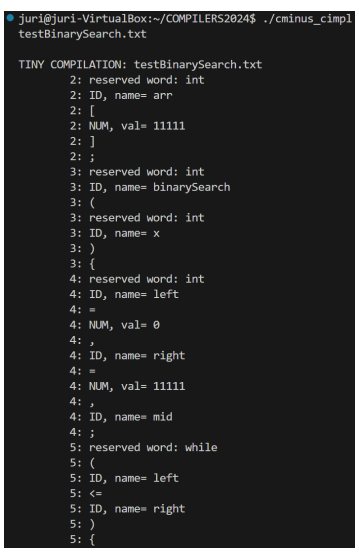


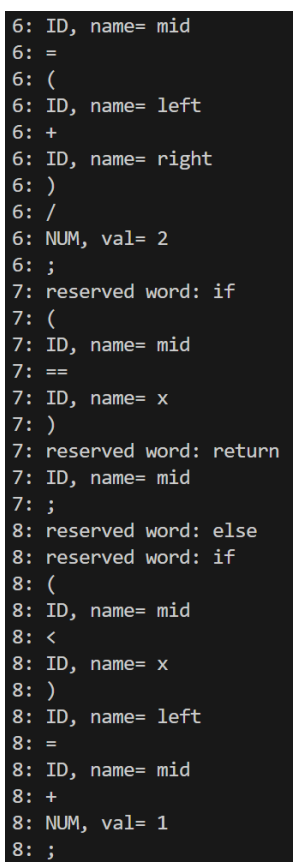


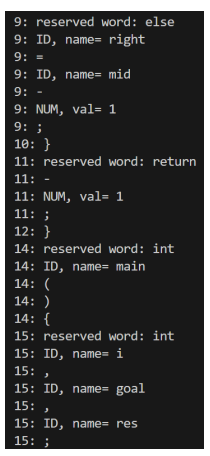
binary search testCase (함수, 배열)

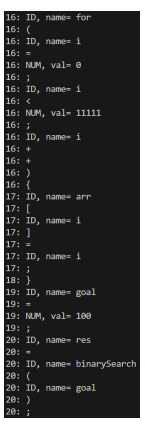
```
/* binary_search program part 1*/
int arr[11111];
int binarySearch(int x) {
   int left=0, right=11111, mid;
   while(left<=right) {
      mid = (left+right)/2;
      if(mid==x) return mid;
      else if(mid<x) left=mid+1;
      else right=mid-1;}
   return -1;
}
```

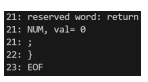
```
/* binary_search program part 2*/
int main() {
   int i, goal, res;
   for(i=0;i<11111;i++) {
      arr[i] = i;
   }
   goal=100;
   res = binarySearch(goal);
   return 0;
}
```











```
juri@juri-VirtualBox:~/COMPILERS2024$ ./cminus_lex  testBinarySearch.txt

TINY COMPILATION: testBinarySearch.txt
        2: reserved word: int
        2: ID, name= arr
        2: [
        2: NUM, val= 11111
        2: ]
        2: ;
        3: reserved word: int
        3: ID, name= binarySearch
        3: (
        3: reserved word: int
        3: ID, name= x
        3: )
        3: {
        4: reserved word: int
        4: ID, name= left
        4: =
        4: NUM, val= 0
        4: ,
        4: ID, name= right
        4: =
        4: NUM, val= 11111
        4: ,
        4: ID, name= mid
        4: ;
```

```
5: reserved word: while
5: (
5: ID, name= left
5: <=
5: ID, name= right
5: )
5: {
6: ID, name= mid
6: =
6: (
6: ID, name= left
6: +
6: ID, name= right
6: )
6: /
6: NUM, val= 2
6: ;
7: reserved word: if
7: (
7: ID, name= mid
7: ==
7: ID, name= x
7: )
7: reserved word: return
7: ID, name= mid
7: ;
```

```
8: reserved word: else
8: reserved word: if
8: (
8: ID, name= mid
8: <
8: ID, name= x
8: )
8: ID, name= left
8: =
8: ID, name= mid
8: +
8: NUM, val= 1
8: ;
9: reserved word: else
9: ID, name= right
9: =
9: ID, name= mid
9: -
9: NUM, val= 1
9: ;
10: }
```

```
11: reserved word: return
11: -
11: NUM, val= 1
11: ;
12: }
14: reserved word: int
14: ID, name= main
14: (
14: )
14: {
15: reserved word: int
15: ID, name= i
15: ,
15: ID, name= goal
15: ,
15: ID, name= res
15: ;
16: ID, name= for
16: (
16: ID, name= i
16: =
16: NUM, val= 0
16: ;
16: ID, name= i
16: <
16: NUM, val= 11111
16: ;
16: ID, name= i
16: +
16: +
16: )
16: {
```

```
17: ID, name= arr
17: [
17: ID, name= i
17: ]
17: =
17: ID, name= i
17: ;
18: }
19: ID, name= goal
19: =
19: NUM, val= 100
19: ;
20: ID, name= res
20: =
20: ID, name= binarySearch
20: (
20: ID, name= goal
20: )
20: ;
21: reserved word: return
21: NUM, val= 0
21: ;
22: }
23: EOF
```

Trouble shooting

- **무한 loop**

  yylex에서 무한 loop를 돌고 EOF를 감지 못 하는 문제가 발생하였다. util.c의 printToken에서는 case를 ENDFILE로 받는데, cminus.l에서 EOF를 만나면 ENDFILE을 반환하게 하는 게 문제였다. return ENDFILE로 변경하니 정상적으로 마무리되었다.

- **whitespace**

  id 3

  id3

  !=

  ! =

  3 3

  이런 상황에서 id 3을 id, num으로 봐야 할 지 공백을 무시하고 하나의 id로 봐야 할 지 고민하다가 공백을 무시한다고 했으니 하나의 id로 보게 코드를 작성했었다. 그러나 lms 질문 게시판에서 letter&letter에서 letter1과 letter2를 다른 token으로 분류하고 싶다면 letter1과 letter2 사이에 white space를 추가해야 된다는 말을 확인했고 코드를 수정하였다.

  각각 {id, num}, {id}, {≠}, {error, =}, {num, num} 으로 인식할 수 있게 INNE, INGT 등 중간 상태 state에서는 공백, 줄바꿈, 탭 등을 만났을 경우의 state를 유지하라는 코드를 제거하였다.

- **comment rule**

  - 주석이 닫히기 전에 EOF를 만나면 error 뜨게 짰는데 인식이 안 되고 무한 루프에 빠졌다

    ⇒ input 결과를 아스키 코드로 출력해보니 0이었고 char로 출력하니 공란이 뜨길래 조건을 0인 경우로 변경해주어서 해결하였다.

  - /* text /* text */ text */ 중첩 comment에서 에러 감지 X

    ⇒ / 이 문자 만나면 input으로 다음에 오는 게 * 이 문자인지 확인하였다. *라면 error 발생.
    이렇게 하면 다음 input을 정확하게 인식하는 데 문제가 있을 것 같아서 ungetToken 함수를 생각했고, 찾아본 결과 lex에서 기본적으로 제공하는 unput 함수가 있다는 것을 알게 되어 해당 함수를 사용하여 해결하였다.

  - 중첩 주석, 주석이 열린 상태에서 EOF 만나는 경우

    ⇒ 위에 작성한 Trouble shooting의 내용대로 코드를 작성했었는데 추후에 이 사항들이 error를 발생하는 사항이 아니었음을 알게 되었다. error 대신 주석 열린 상태로 EOF 만나면 그대로 EOF 출력 후 종료, 중첩 주석의 경우 ex) /* /* */ */ 두 번째 */에서 주석이 끝나고 *와 /는 tokenizing을 하는 걸로 코드 구조를 변경하였다.