1. 문서화주석

- /** ~ */

- 문서화 주석의 경우에는 주석의 내용을 출력해야 함.

구현 :

```
case '/': // 주석 처리 부분
                        ch = fgetc(sourceFile);
                        if (ch == '*') {
                                ch = fgetc(sourceFile);
                                if (ch == '*')
                                {
                                        ch = fgetc(sourceFile);
                                        do {//문서화 주석
                                                while (ch != '*')
                                                {
                                                        commentArr[j++] = ch;
                                                        ch = fgetc(sourceFile);
                                                }
                                                commentArr[j] = '₩0';
                                                ch = fgetc(sourceFile);
                                        } while (ch != '/');
                                        printf("₩n문서화 주석 : %s₩n", commentArr); //문서화
주석 출력
                                }
```

Fgetc로 읽어 들인 문자열의 형태가 /**이면 내부 do-while문 안에서 /** 이후의 문자를 commentArr 배열에 저장합니다. commentArr 배열은 문서화 주석 시에 주석 내용을 출력하기 위해 사용되는 배열입니다. do-while문 내부의 while문이 '*'를 인식할 때까지 문자를 읽어서 commentArr 배열에 저장하고 '*" 인식되면 while문이 종료됩니다. 그런 다음 commentArr에 저장된 문자열의 끝을 지정해주기 위해서 '₩0'을 삽입해줍니다. 마지막으로 '/' 인식되면 문서화 주석을 읽어 들이는 반복문이 끝나게 됩니다. 그런 다음 문서화 주석의 내용을 printf로 출력했습니다.

2. Double Literal

- 고정소숫점, 부동소숫점

- 요약형식: 0. , .0 , 단, . 만 나오는 것은 인식하지 말 것

아래 코드는 스캐너가 고정소숫점을 인식하는 코드입니다.

```
else if (isdigit(ch)) {  // number or floating number
                        int i;

                        //소수인지 확인하기 위한 for문
                        //숫자에 '.' 이 나오면 소수, 소수인 경우 fBool 변수는 TRUE로 초기화
                        //정수는 fBool 변수 FALSE로 초기화
                        for (i = 10; i > 0; i--)
                        {
```

```
                if (ch == '.')
                {
                        fBool = TRUE;
                        break;
                }
                else if (ch == ';')
                        break;

                ch = fgetc(sourceFile);
                k++;
        }
        k++;

        //소수 판단을 위해서 이동한 파일 포인터를 원상태로 복원
        fseek(sourceFile, -k, 1);

        if (fBool == FALSE) //정수
        {
                token.number = tnumber;
                token.value.num = getNumber(ch);
        }
        else //소수
        {
                token.number = tFnumber;
                token.fNumber = getFnumber();
        }

}
```

기존 정수를 인식하는 else if 조건 문에 소수를 인식하기 위해 코드를 추가했습니다. 우선 처음에 for문을 사용해서 숫자에 '.' 이 나오는지 확인합니다. 숫자안에 '.' 이 존재하면 소수임을 인식하고 소수 여부 확인 변수인 fBool을 TRUE로 초기화합니다. fBool 변수는 생성될 시에 디폴트로 FALSE로 초기화됩니다. 그런 다음에 소수임을 판단하기 위해 이동된 파일 포인터를 숫자의 처음 부분으로 되돌려 주기 위해서 fseek 함수를 사용했습니다. 현재 위치에서 -k 만큼 좌측 방향으로 sourceFile의 파일 포인터를 이동한 후 fBool의 저장된 조건에 따라서 정수 또는 소수 인식 부분으로 넘어갑니다. fBool이 FALSE이면 정수임을 나타내고 if문 안에서 getNumber 함수를 사용해서 정수를 토큰 형태로 가져옵니다. fBool이 TRUE이면 소수임을 나타내고 else문 안에서 getFnumber 함수를 사용해서 소수를 토큰 형태로 가져옵니다. 아래 코드는 getFnumber 함수의 코드입니다.

```
double getFnumber()
{
        char ch;
        char tFnumArr[20];
        int i = 0;

        memset(tFnumArr, 0, sizeof(tFnumArr));

        ch = fgetc(sourceFile);
        while (1) //소수 인식
        {
                if (ch == ';' || ch == ',')
```

```
                {
                        ungetc(ch, sourceFile);
                        break;
                }
                tFnumArr[i++] = ch;
                ch = fgetc(sourceFile);
        }

        return atof(tFnumArr);//문자열을 실수로 변환해서 반환
}
```

getFnumber가 호출되기 직전에 파일 포인터를 숫자의 처음으로 이동했기 때문에 fgetc를 사용해서 문자 숫자를 읽습니다. 문자 숫자가 숫자의 끝을 의미하는 ';' 또는 ','을 인식할 때까지 숫자를 읽어서 tFnumArr 임시배열에 저장했습니다. 만약 숫자의 끝을 의미하는 ';' 또는 ',' 이 인식되면 while문이 종료되고 tFnumArr에 저장된 문자 숫자를 atof 함수를 사용해서 고정소수점으로 변환한 후 반환했습니다. 아래 코드는 소수를 출력하는 부분으로 printToken 함수의 코드 내부에 아래 코드를 추가했습니다.

```
else if (token.number == tFnumber)
                printf("number: %d, value: %f, value: %e₩n", token.number, token.fNumber, token.fNumber);
```

토큰의 번호가 tFnumber이면 소수임을 나타내고 소수를 고정소수점 방식과 부동소수점 방식으로 출력했습니다. tFnumber 값은 tsymbol에서 61로 저장되었습니다.

3. string literal - clear

- C 언어의 이스케이프 시퀀스 포함

- 참고: https://en.wikipedia.org/wiki/Escape_sequences_in_C

- 단 유니코드 이스케이프 시퀀스는 제외

이스케이프 시퀀스를 인식하기 위해서 이스케이프 시퀀스를 tokenName 배열에 저장했습니다. 아래 코드는 이스케이프 시퀀스가 추가된 tokenName에 정보입니다.

```
char *tokenName[] = {
        "!",        "!=",       "%",        "%=",       "%ident",    "%number",
        /* 0          1           2           3           4            5          */
        "&&",       "(",        ")",        "*",        "*=",        "+",
        /* 6          7           8           9           10           11         */
        "++",       "+=",       ",",        "-",        "--",       "-=",
        /* 12         13          14          15          16           17         */
        "/",        "/=",       ";",        "<",        "<=",        "=",
        /* 18         19          20          21          22           23         */
        "==",       ">",        ">=",       "[",        "]",         "eof",
        /* 24         25          26          27          28           29         */
        //   ...........    word symbols ..................................... //
        /* 30         31          32          33          34           35         */
        "const",    "else",     "if",       "int",      "return",   "void",
        /* 36         37          38          39          40                      */
```

```
        "while",    "{",        "||",       "}",      "for",
        /* 41         42         43         44         45          46       */
        "switch",   "case",     "goto",    "break",   "continue",   "double",
        /* 47       48        49        50         51        52              */
        ":",    "₩₩a",    "₩₩b",   "₩₩f",   "₩₩n",    "₩₩r",
        /*53     54     55     56    57    58*/
        "₩₩t",   "₩₩v",    "₩₩₩₩",    "₩₩₩'",    "₩₩₩"",    "₩₩?",
        /* 59   60*/
        "₩₩e",  "₩""
};
```

이스케이프 시퀀스는 48번부터 59번까지입니다. 이스케이프 시퀀스에 대한 토큰 넘버를 저장하기 위해서 Scanner.h의 tsymbol에도 이스케이프 시퀀스를 추가했습니다. 아래 코드는 이스케이프 시퀀스가 추가된 tsymbol입니다.

```
enum tsymbol {
        tnull = -1,
        tnot, tnotequ, tremainder, tremAssign, tident, tnumber,
        /* 0          1           2         3           4          5     */
        tand, tlparen, trparen, tmul, tmulAssign, tplus,
        /* 6          7           8         9           10         11    */
        tinc, taddAssign, tcomma, tminus, tdec, tsubAssign,
        /* 12         13          14        15          16         17    */
        tdiv, tdivAssign, tsemicolon, tless, tlesse, tassign,
        /* 18         19          20        21          22         23    */
        tequal, tgreat, tgreate, tlbracket, trbracket, teof,
        /* 24         25          26        27          28         29    */
        //   ...........    word symbols ................................ //
        /* 30         31          32        33          34         35    */
        tconst, telse, tif, tint, treturn, tvoid,
        /* 36         37          38        39        40                    */
        twhile, tlbrace, tor, trbrace, tfor,
        /*41          42          43        44        45      46            */
        tswitch, tcase, tgoto, tbreak, tcontinue, tdouble,
        //---------------------escape sequence, :, " -------------------------
        /* 47       48        49        50         51        52             */
         tdelimiter, tescapeA, tescapeB, tescapeF, tescapeN, tescapeR,
        /* 53     54     55     56    57    58                              */
        tescapeT, tescapeV, tescapeBack, tescapeSingle, tescapeDouble, tescapeQuestion,
        /* 59    60    61*/
        tescapeE, tdoublequote, tFnumber
};
```

이스케이프 시퀀스는 printf 의 double quote 안에서 인식됩니다. 그러므로 double quote를 scanner에서 인식하기 위해서 60번에 tdoublequote도 추가했습니다. 아래 코드는 이스케이프 시퀀스를 인식하는 Scanner.cpp 내부의 코드입니다.

```
case'₩₩': //이스케이프 시퀀스 인식
                        ch = fgetc(sourceFile);
                        if (ch == 'a') token.number = tescapeA;
                        else if (ch == 'b') token.number = tescapeB;
                        else if (ch == 'f') token.number = tescapeF;
                        else if (ch == 'n') token.number = tescapeN;
```

```
else if (ch == 'r') token.number = tescapeR;
else if (ch == 't') token.number = tescapeT;
else if (ch == 'v') token.number = tescapeV;
else if (ch == '₩₩') token.number = tescapeBack;
else if (ch == '₩'')token.number = tescapeSingle;
else if (ch == '₩"')token.number = tescapeDouble;
else if (ch == '₩?')token.number = tescapeQuestion;
else if (ch == 'e') token.number = tescapeE;
break;
```

switch-case문에서 백슬래쉬가 인식되면 이스케이프 시퀀스가 입력된다는 것을 의미합니다. 그러므로 조건을 case '₩₩'로 구성하고 백슬래쉬 다음에 문자를 fgetc로 읽어옵니다. 읽어온 문자에 따라 if-elseif 조건문을 사용해서 토큰에 대한 번호를 token 구조체의 int number 변수에 저장했습니다. 저장된 토큰은 아래 코드에서 출력합니다.

```
void printToken(struct tokenType token)
{
        if (token.number == tident)
                printf("number: %d, value: %s₩n", token.number, token.value.id);
        else if (token.number == tnumber)
                printf("number: %d, value: %d₩n", token.number, token.value.num);
        else if (token.number == tFnumber)
                printf("number: %d, value: %f, value: %e₩n", token.number, token.fNumber,
token.fNumber);
        else
                printf("number: %d(%s)₩n", token.number, tokenName[token.number]);

}
```

Token.number는 이스케이프 시퀀스를 의미하는 토큰 번호로 저장되었기 때문에 마지막 else문에서 이스케이프 시퀀스가 출력됩니다.

4. 키워드 추가

- for, switch, case, goto, break, continue, double

새로운 키워드인 for, switch, case, goto, break, continue, double을 인식하기 위해서 아래의 코드를 수정했습니다.

```
char *keyword[NO_KEYWORD] = {
        "const",  "else",    "if",    "int",    "return",  "void",    "while",
        "for", "switch", "case", "goto", "break", "continue", "double"
};

enum tsymbol tnum[NO_KEYWORD] = {
        tconst,    telse,     tif,     tint,     treturn,   tvoid,     twhile,
        tfor, tswitch, tcase, tgoto, tbreak, tcontinue, tdouble
};
```

토큰의 키워드를 검색하는 코드는 아래와 같습니다.

```
                                                // find the identifier in the keyword table
```

```
for (index = 0; index < NO_KEYWORD; index++)
        if (!strcmp(id, keyword[index])) break;
if (index < NO_KEYWORD)    // found, keyword exit
        token.number = tnum[index];
else {                     // not found, identifier exit
        token.number = tident;
        strcpy_s(token.value.id, id);
}
```

For문 내부에서 입력된 문자열을 가지고 keyword 배열을 검색합니다. 입력된 문자열이 keyword 배열의 원소 중에서 일치하는 원소가 존재하면 반복문을 종료하고 원소의 index 번호를 가지고 enum tsymbol 자료형의 tnum 배열에서 토큰의 번호를 가져옵니다. 만약 입력된 문자열이 keyword 배열에 문자열이 존재하지 않으면 token.number를 tident로 설정합니다. Tsymbol, tokenName 배열에서 새로 추가된 키워드는 40~46번에 존재합니다.

5. 구분자 추가

- :

: 구분자를 Scanner.cpp의 tokenName[]에 추가했습니다. 또한 토큰 번호를 얻기 위해서 Scanner.h의 tsymbol에도 : 구분자를 추가했습니다. : 추가된 코드는 3번 문제 이스케이프 시퀀스에 추가한 tokenName, tsymbol 코드와 동일해서 여기서는 생략했습니다. 아래 코드는 Scanner.cpp 내부에서 : 와 "를 인식하는 코드입니다.

```
case ':': token.number = tdelimiter;      break; // : 구분자 추가
case '₩"': token.number = tdoublequote;   break; // : double quote 추가
```

case문을 사용해서 : 이 검색되면 : 에 해당하는 토큰 번호를 token.number에 저장합니다. printf 내부에 문자열을 나타날 때 사용하는 double quote를 인식하는 case문도 추가했습니다. Tsymbol, tokenName 안에서 : 는 47번에 "는 60번에 저장되어 있습니다.

결과 :

아래 코드는 mini C scanner의 새로 추가한 토큰을 확인하기 위해 임의로 작성한 코드입니다.

```
/*
    Mini C has two types of comments: text comment and line comment.
*/
/**
    문서화 주석의 경우 내용을 출력해야 합니다.
    문서화 주석 부분입니다.
*/
void main()
{
  printf("₩a ₩b ₩f ₩n ₩r ₩t ₩v ₩₩ ₩' ₩" ₩? ₩e");
```

```
  JUMP:
  int i;
  double d=100.5236;
  i = 1000;         // i sets 100
  write(i);         // printf it
  for( ; ;);
  switch(){
    case : break;
  }
  while(){
    d=0.333;
    continue;
  }
  goto JUMP;
}
```

결과 화면 :

```
*** start of Mini C Compiler
  * source file name: example.mc
=== start of Scanner

문서화 주석 :
    문서화 주석의 경우 내용을 출력해야 합니다.
    문서화 주석 부분입니다.

Current Token --> number: 35(void)
Current Token --> number: 4, value: main
Current Token --> number: 7(()
Current Token --> number: 8())
Current Token --> number: 37({)
Current Token --> number: 4, value: printf
Current Token --> number: 7(()
Current Token --> number: 60(")
Current Token --> number: 48(₩a)
Current Token --> number: 49(₩b)
Current Token --> number: 50(₩f)
Current Token --> number: 51(₩n)
Current Token --> number: 52(₩r)
Current Token --> number: 53(₩t)
Current Token --> number: 54(₩v)
Current Token --> number: 55(₩₩)
Current Token --> number: 56(₩')
Current Token --> number: 57(₩")
Current Token --> number: 58(₩?)
Current Token --> number: 59(₩e)
Current Token --> number: 60(")
Current Token --> number: 8())
Current Token --> number: 20(;)
Current Token --> number: 4, value: JUMP
Current Token --> number: 47(:)
Current Token --> number: 33(int)
Current Token --> number: 4, value: i
Current Token --> number: 20(;)
Current Token --> number: 46(double)
Current Token --> number: 4, value: d
Current Token --> number: 23(=)
Current Token --> number: 61, value: 100.523600, value: 1.005236e+02
```

```
Current Token --> number: 20(;)
Current Token --> number: 4, value: i
Current Token --> number: 23(=)
Current Token --> number: 5, value: 1000
Current Token --> number: 20(;)
Current Token --> number: 4, value: write
Current Token --> number: 7(()
Current Token --> number: 4, value: i
Current Token --> number: 8())
Current Token --> number: 20(;)
Current Token --> number: 40(for)
Current Token --> number: 7(()
Current Token --> number: 20(;)
Current Token --> number: 20(;)
Current Token --> number: 8())
Current Token --> number: 20(;)
Current Token --> number: 41(switch)
Current Token --> number: 7(()
Current Token --> number: 8())
Current Token --> number: 37({)
Current Token --> number: 42(case)
Current Token --> number: 47(:)
Current Token --> number: 44(break)
Current Token --> number: 20(;)
Current Token --> number: 39(})
Current Token --> number: 36(while)
Current Token --> number: 7(()
Current Token --> number: 8())
Current Token --> number: 37({)
Current Token --> number: 4, value: d
Current Token --> number: 23(=)
Current Token --> number: 61, value: 0.333000, value: 3.330000e-01
Current Token --> number: 20(;)
Current Token --> number: 45(continue)
Current Token --> number: 20(;)
Current Token --> number: 39(})
Current Token --> number: 43(goto)
Current Token --> number: 4, value: JUMP
Current Token --> number: 20(;)
Current Token --> number: 39(})
```