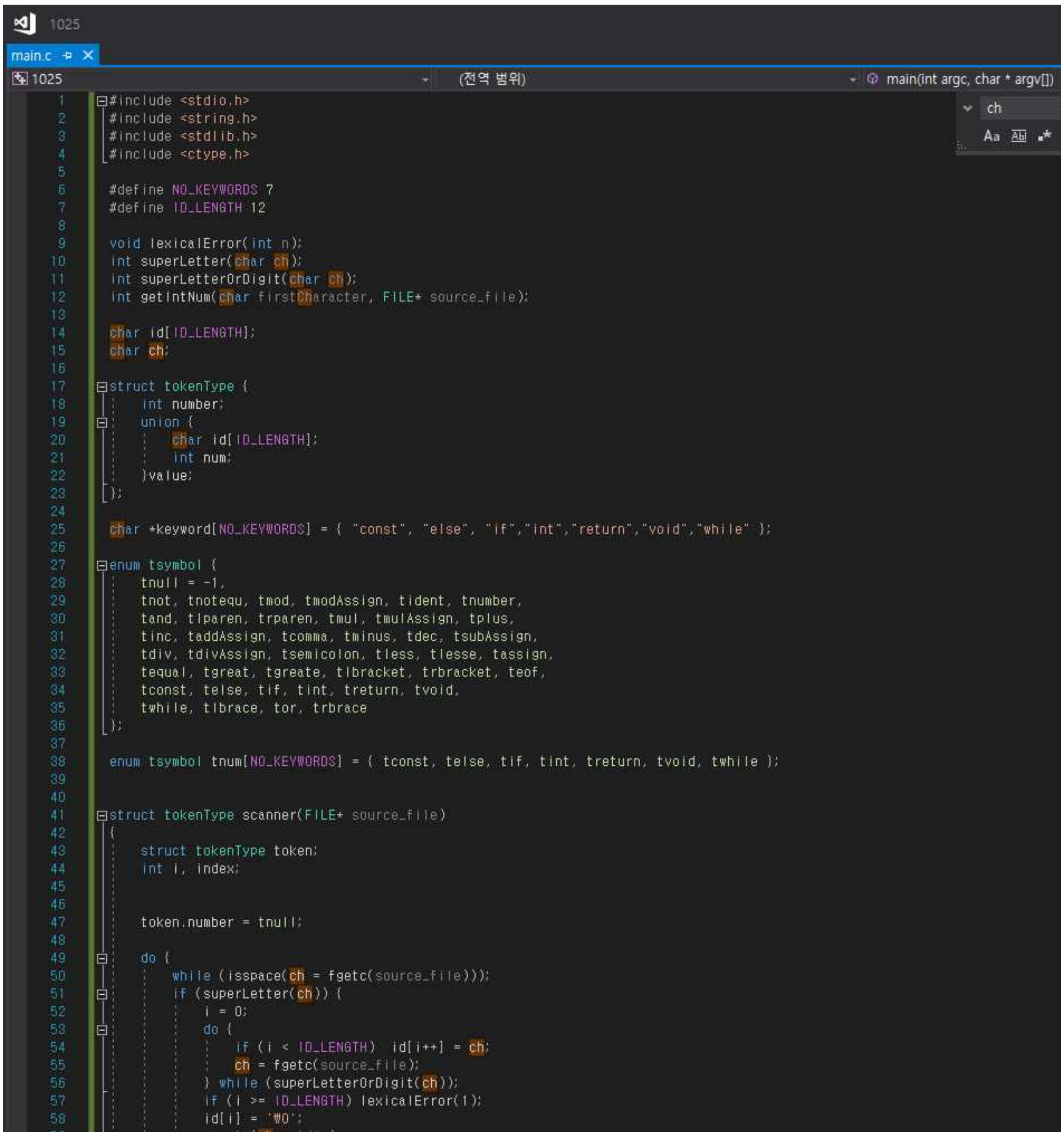
REPORT

“ CH04 Scanner 구현 ”

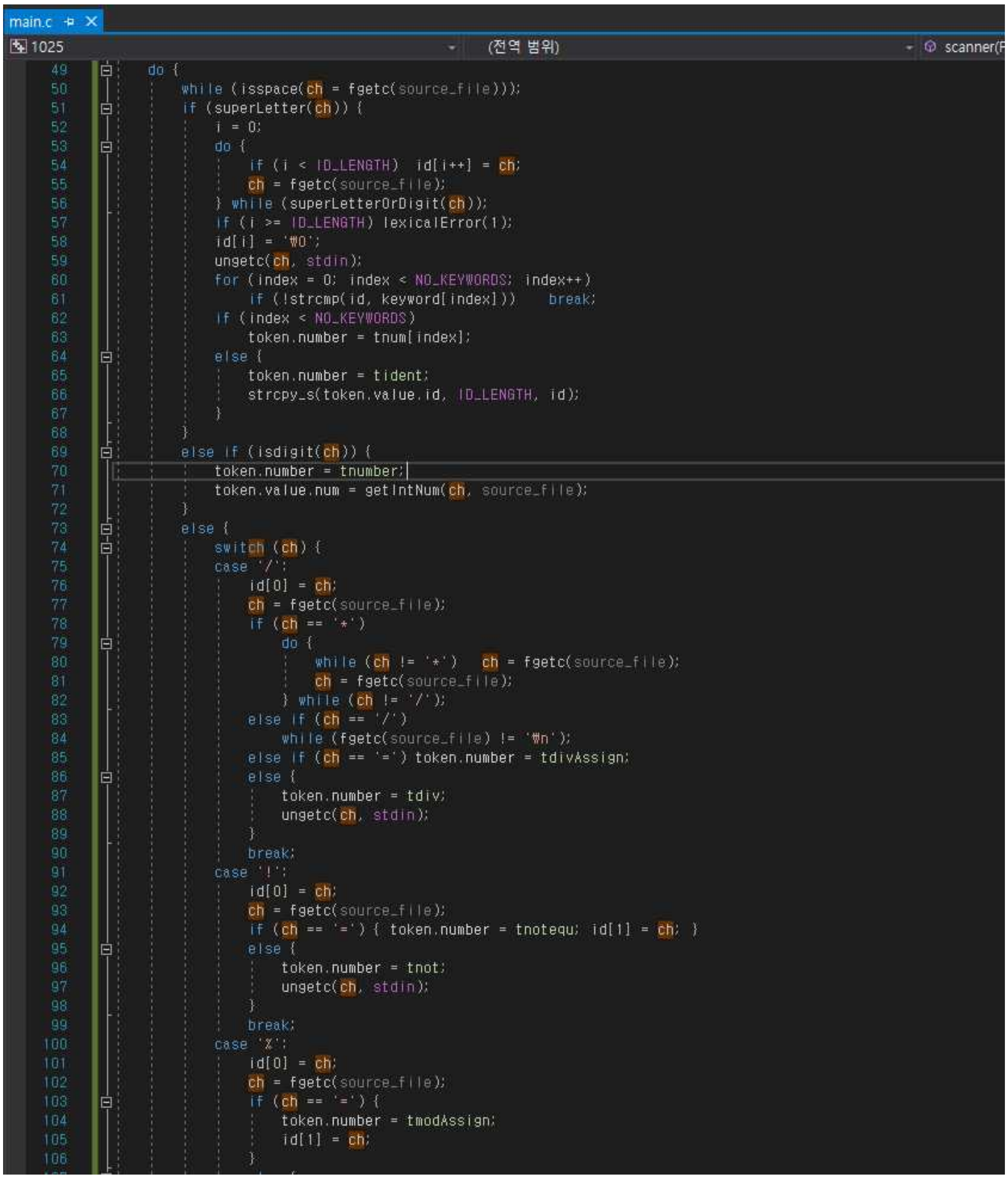
과 목 명	컴파일러 설계 및 구축 (월25,26)
담당교수	이양선 교수님
학 과	컴퓨터공학과
학 번	2015305084
이 름	홍송희
제 출 일	2018.10.25

4.11 본문에 있는 함수 scanner()를 이용하여 실질적인 Mini C의 어휘 분석기를 구현하시오. 이때, 입력 프로그램은 부록 A에 있는 prime.mc를 사용하고 출력의 형태는 다음과 같은 형태로 하시오.

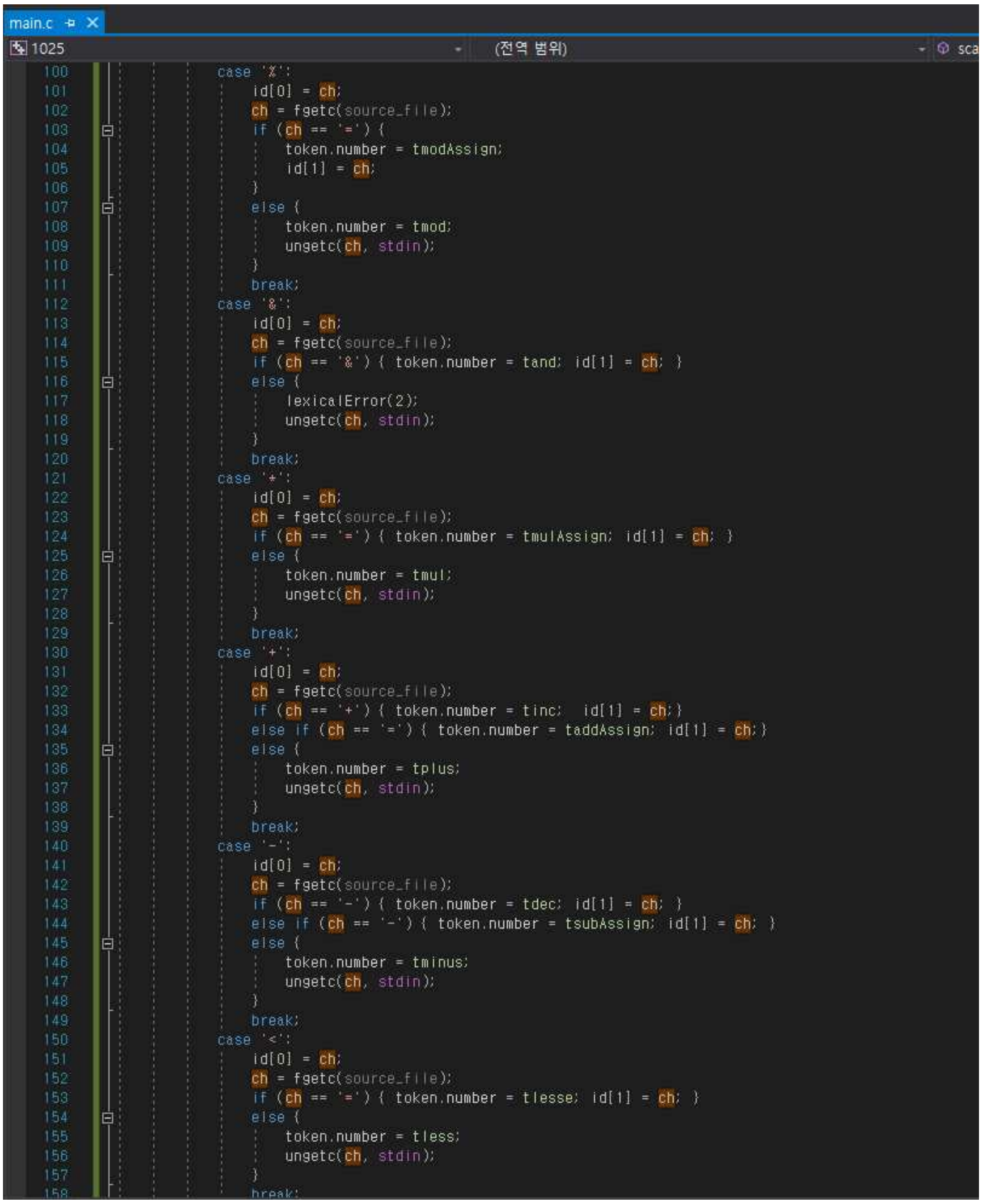
<소스 코드>



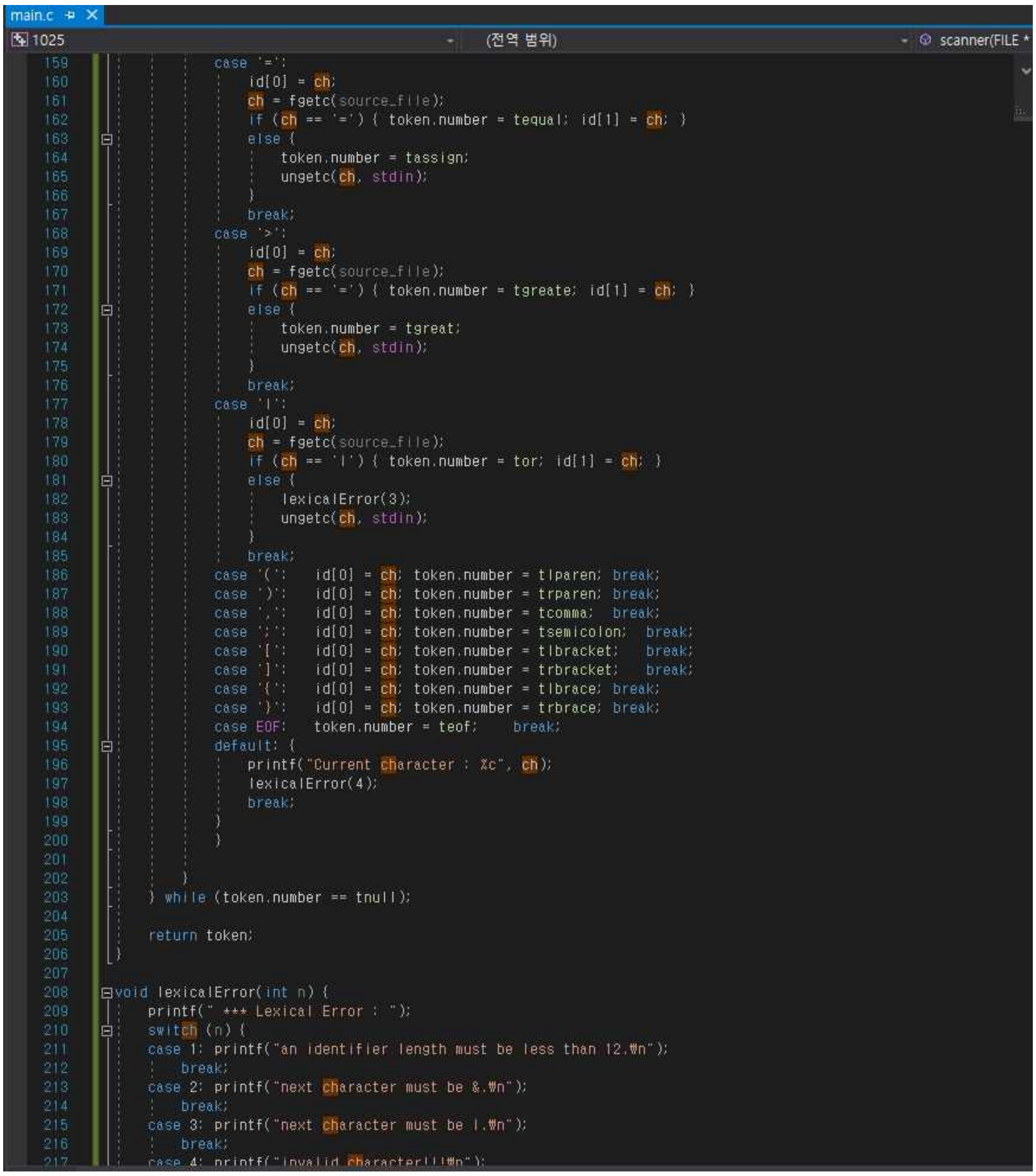
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define NO_KEYWORDS 7
#define ID_LENGTH 12

void lexicalError(int n);
int superLetter(char ch);
int superLetterOrDigit(char ch);
int getIntNum(char firstCharacter, FILE* source_file);

char id[ID_LENGTH];
char ch;

struct tokenType {
	int number;
	union {
		char id[ID_LENGTH];
		int num;
	}value;
};

char *keyword[NO_KEYWORDS] = { "const", "else", "if","int","return","void","while" };

enum tsymbol {
	tnull = -1,
	tnot, tnotequ, tmod, tmodAssign, tident, tnumber,
	tand, tlparen, trparen, tmul, tmulAssign, tplus,
	tinc, taddAssign, tcomma, tminus, tdec, tsubAssign,
	tdiv, tdivAssign, tsemicolon, tless, tlesse, tassign,
	tequal, tgreat, tgreate, tlbracket, trbracket, teof,
	tconst, telse, tif, tint, treturn, tvoid,
	twhile, tlbrace, tor, trbrace
};

enum tsymbol tnum[NO_KEYWORDS] = { tconst, telse, tif, tint, treturn, tvoid, twhile };


struct tokenType scanner(FILE* source_file)
{
	struct tokenType token;
	int i, index;


	token.number = tnull;

	do {
		while (isspace(ch = fgetc(source_file)));
		if (superLetter(ch)) {
			i = 0;
			do {
				if (i < ID_LENGTH)  id[i++] = ch;
				ch = fgetc(source_file);
			} while (superLetterOrDigit(ch));
			if (i >= ID_LENGTH) lexicalError(1);
			id[i] = '₩0';
```



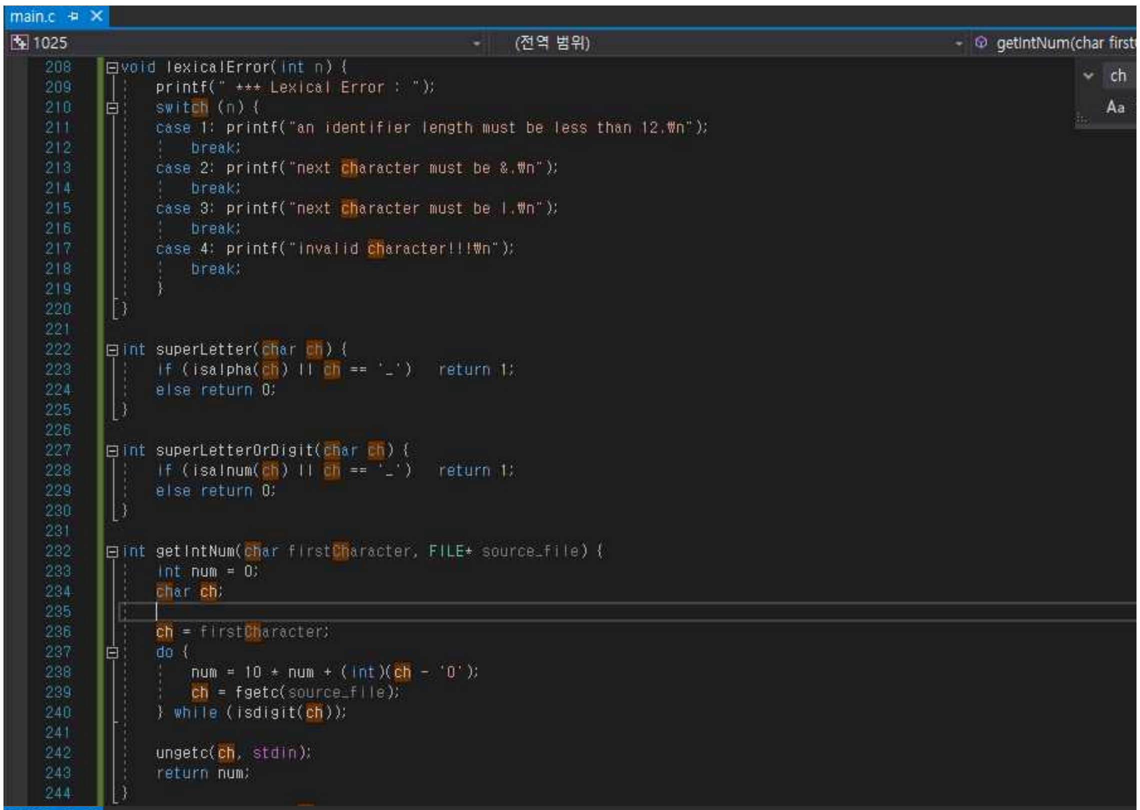
```c
    do {
        while (isspace(ch = fgetc(source_file)));
        if (superLetter(ch)) {
            i = 0;
            do {
                if (i < ID_LENGTH)  id[i++] = ch;
                ch = fgetc(source_file);
            } while (superLetterOrDigit(ch));
            if (i >= ID_LENGTH) lexicalError(1);
            id[i] = '₩0';
            ungetc(ch, stdin);
            for (index = 0; index < NO_KEYWORDS; index++)
                if (!strcmp(id, keyword[index]))    break;
            if (index < NO_KEYWORDS)
                token.number = tnum[index];
            else {
                token.number = tident;
                strcpy_s(token.value.id, ID_LENGTH, id);
            }
        }
        else if (isdigit(ch)) {
            token.number = tnumber;
            token.value.num = getIntNum(ch, source_file);
        }
        else {
            switch (ch) {
            case '/':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '*')
                    do {
                        while (ch != '*')   ch = fgetc(source_file);
                        ch = fgetc(source_file);
                    } while (ch != '/');
                else if (ch == '/')
                    while (fgetc(source_file) != '₩n');
                else if (ch == '=') token.number = tdivAssign;
                else {
                    token.number = tdiv;
                    ungetc(ch, stdin);
                }
                break;
            case '!':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '=') { token.number = tnotequ; id[1] = ch; }
                else {
                    token.number = tnot;
                    ungetc(ch, stdin);
                }
                break;
            case '%':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '=') {
                    token.number = tmodAssign;
                    id[1] = ch;
                }
```



```c
            case '%':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '=') {
                    token.number = tmodAssign;
                    id[1] = ch;
                }
                else {
                    token.number = tmod;
                    ungetc(ch, stdin);
                }
                break;
            case '&':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '&') { token.number = tand; id[1] = ch; }
                else {
                    lexicalError(2);
                    ungetc(ch, stdin);
                }
                break;
            case '*':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '=') { token.number = tmulAssign; id[1] = ch; }
                else {
                    token.number = tmul;
                    ungetc(ch, stdin);
                }
                break;
            case '+':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '+') { token.number = tinc;  id[1] = ch;}
                else if (ch == '=') { token.number = taddAssign; id[1] = ch;}
                else {
                    token.number = tplus;
                    ungetc(ch, stdin);
                }
                break;
            case '-':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '-') { token.number = tdec; id[1] = ch; }
                else if (ch == '-') { token.number = tsubAssign; id[1] = ch; }
                else {
                    token.number = tminus;
                    ungetc(ch, stdin);
                }
                break;
            case '<':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '=') { token.number = tlesse; id[1] = ch; }
                else {
                    token.number = tless;
                    ungetc(ch, stdin);
                }
                break;
```



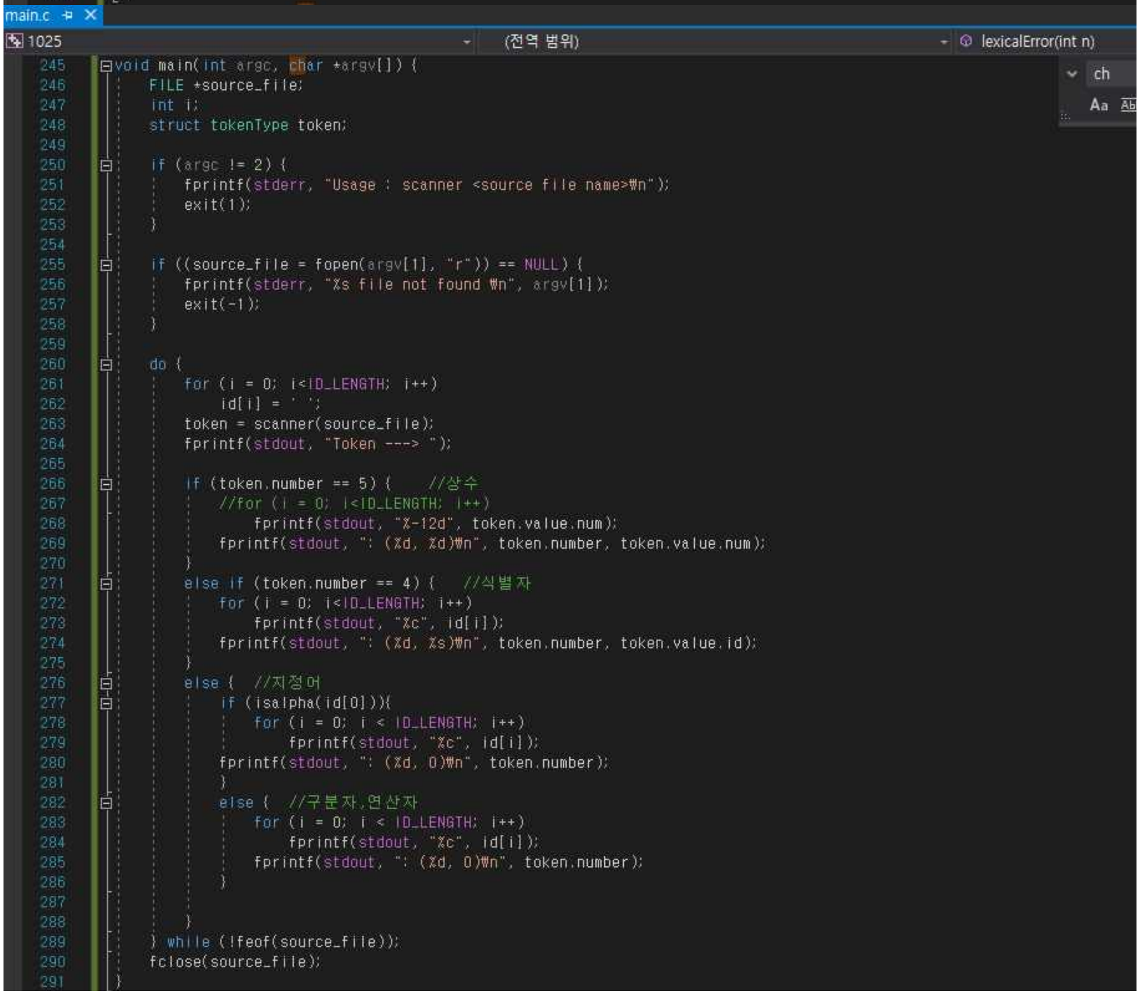
```c
                case '=':
                    id[0] = ch;
                    ch = fgetc(source_file);
                    if (ch == '=') { token.number = tequal; id[1] = ch; }
                    else {
                        token.number = tassign;
                        ungetc(ch, stdin);
                    }
                    break;
                case '>':
                    id[0] = ch;
                    ch = fgetc(source_file);
                    if (ch == '=') { token.number = tgreate; id[1] = ch; }
                    else {
                        token.number = tgreat;
                        ungetc(ch, stdin);
                    }
                    break;
                case '|':
                    id[0] = ch;
                    ch = fgetc(source_file);
                    if (ch == '|') { token.number = tor; id[1] = ch; }
                    else {
                        lexicalError(3);
                        ungetc(ch, stdin);
                    }
                    break;
                case '(':   id[0] = ch; token.number = tlparen; break;
                case ')':   id[0] = ch; token.number = trparen; break;
                case ',':   id[0] = ch; token.number = tcomma;  break;
                case ';':   id[0] = ch; token.number = tsemicolon;  break;
                case '[':   id[0] = ch; token.number = tlbracket;   break;
                case ']':   id[0] = ch; token.number = trbracket;   break;
                case '{':   id[0] = ch; token.number = tlbrace; break;
                case '}':   id[0] = ch; token.number = trbrace; break;
                case EOF:   token.number = teof;    break;
                default: {
                    printf("Current character : %c", ch);
                    lexicalError(4);
                    break;
                }
                }

            }
    } while (token.number == tnull);

    return token;
}

void lexicalError(int n) {
    printf(" *** Lexical Error : ");
    switch (n) {
    case 1: printf("an identifier length must be less than 12.₩n");
        break;
    case 2: printf("next character must be &.₩n");
        break;
    case 3: printf("next character must be |.₩n");
        break;
    case 4: printf("invalid character!!₩n");
```



main.c

1025 (전역 범위) getIntNum(char first

```c
void lexicalError(int n) {
    printf(" *** Lexical Error : ");
    switch (n) {
    case 1: printf("an identifier length must be less than 12.₩n");
        break;
    case 2: printf("next character must be &.₩n");
        break;
    case 3: printf("next character must be |.₩n");
        break;
    case 4: printf("invalid character!!!₩n");
        break;
    }
}

int superLetter(char ch) {
    if (isalpha(ch) || ch == '_')   return 1;
    else return 0;
}

int superLetterOrDigit(char ch) {
    if (isalnum(ch) || ch == '_')   return 1;
    else return 0;
}

int getIntNum(char firstCharacter, FILE* source_file) {
    int num = 0;
    char ch;

    ch = firstCharacter;
    do {
        num = 10 * num + (int)(ch - '0');
        ch = fgetc(source_file);
    } while (isdigit(ch));

    ungetc(ch, stdin);
    return num;
}
```

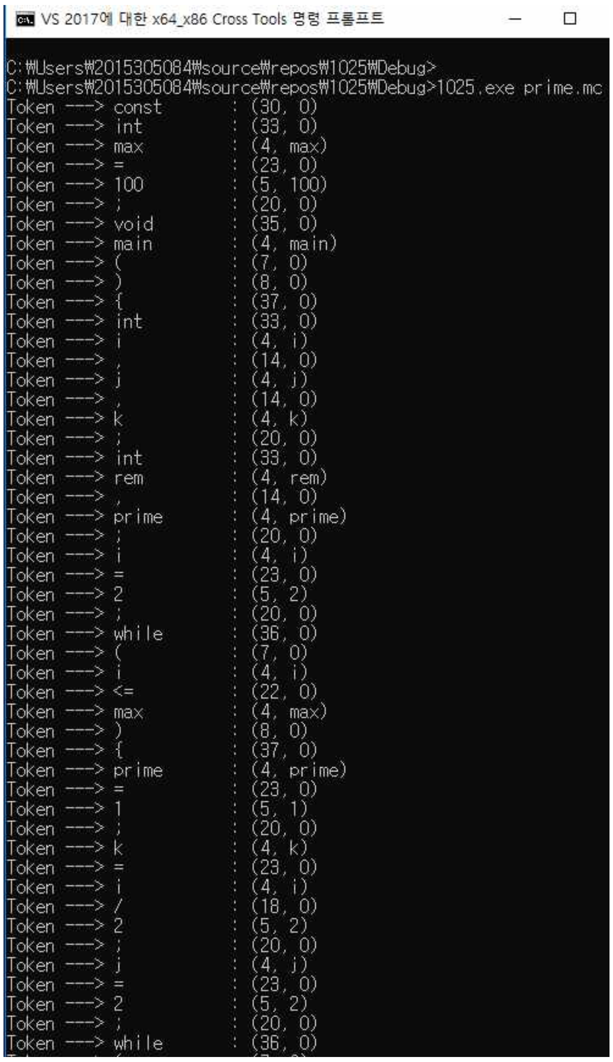


main.c

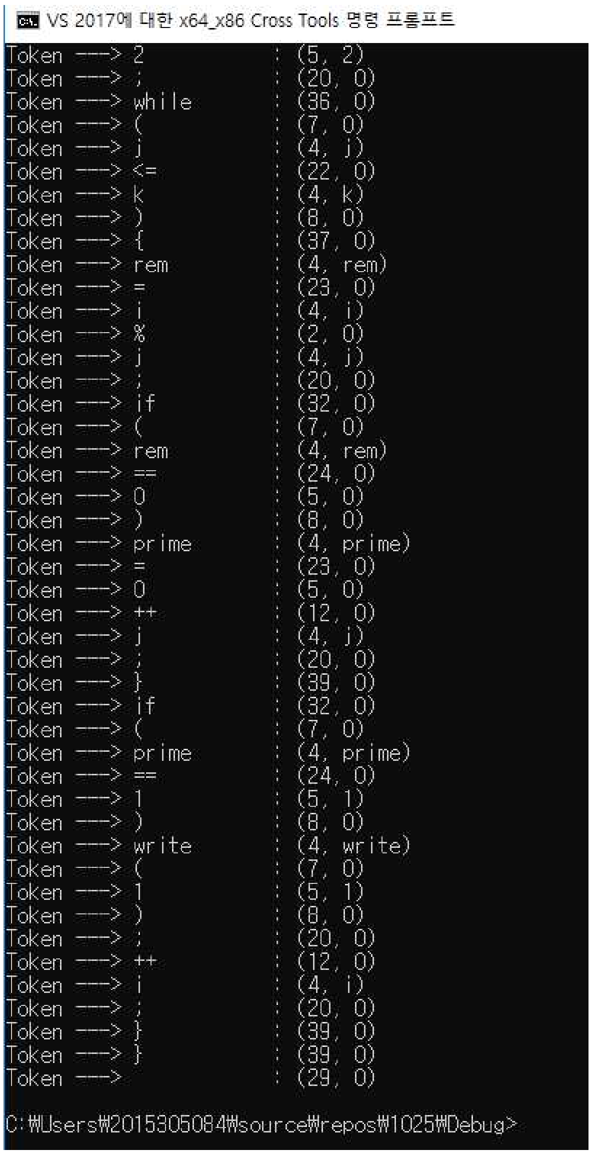
1025 (전역 범위) lexicalError(int n)

```c
void main(int argc, char *argv[]) {
    FILE *source_file;
    int i;
    struct tokenType token;

    if (argc != 2) {
        fprintf(stderr, "Usage : scanner <source file name>₩n");
        exit(1);
    }

    if ((source_file = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "%s file not found ₩n", argv[1]);
        exit(-1);
    }

    do {
        for (i = 0; i<ID_LENGTH; i++)
            id[i] = ' ';
        token = scanner(source_file);
        fprintf(stdout, "Token ---> ");

        if (token.number == 5) {    //상수
            //for (i = 0; i<ID_LENGTH; i++)
                fprintf(stdout, "%-12d", token.value.num);
            fprintf(stdout, ": (%d, %d)₩n", token.number, token.value.num);
        }
        else if (token.number == 4) {   //식별자
            for (i = 0; i<ID_LENGTH; i++)
                fprintf(stdout, "%c", id[i]);
            fprintf(stdout, ": (%d, %s)₩n", token.number, token.value.id);
        }
        else {  //지정어
            if (isalpha(id[0])){
                for (i = 0; i < ID_LENGTH; i++)
                    fprintf(stdout, "%c", id[i]);
            fprintf(stdout, ": (%d, 0)₩n", token.number);
            }
            else {  //구분자,연산자
                for (i = 0; i < ID_LENGTH; i++)
                    fprintf(stdout, "%c", id[i]);
                fprintf(stdout, ": (%d, 0)₩n", token.number);
            }

        }
    } while (!feof(source_file));
    fclose(source_file);
}
```

<실행결과>



```
C:\Users\2015305084\source\repos\1025\Debug>
C:\Users\2015305084\source\repos\1025\Debug>1025.exe prime.mc
Token ---> const         : (30, 0)
Token ---> int           : (33, 0)
Token ---> max           : (4, max)
Token ---> =             : (23, 0)
Token ---> 100           : (5, 100)
Token ---> ;             : (20, 0)
Token ---> void          : (35, 0)
Token ---> main          : (4, main)
Token ---> (             : (7, 0)
Token ---> )             : (8, 0)
Token ---> {             : (37, 0)
Token ---> int           : (33, 0)
Token ---> i             : (4, i)
Token ---> ,             : (14, 0)
Token ---> j             : (4, j)
Token ---> ,             : (14, 0)
Token ---> k             : (4, k)
Token ---> ;             : (20, 0)
Token ---> int           : (33, 0)
Token ---> rem           : (4, rem)
Token ---> ,             : (14, 0)
Token ---> prime         : (4, prime)
Token ---> ;             : (20, 0)
Token ---> i             : (4, i)
Token ---> =             : (23, 0)
Token ---> 2             : (5, 2)
Token ---> ;             : (20, 0)
Token ---> while         : (36, 0)
Token ---> (             : (7, 0)
Token ---> i             : (4, i)
Token ---> <=            : (22, 0)
Token ---> max           : (4, max)
Token ---> )             : (8, 0)
Token ---> {             : (37, 0)
Token ---> prime         : (4, prime)
Token ---> =             : (23, 0)
Token ---> 1             : (5, 1)
Token ---> ;             : (20, 0)
Token ---> k             : (4, k)
Token ---> =             : (23, 0)
Token ---> i             : (4, i)
Token ---> /             : (18, 0)
Token ---> 2             : (5, 2)
Token ---> ;             : (20, 0)
Token ---> j             : (4, j)
Token ---> =             : (23, 0)
Token ---> 2             : (5, 2)
Token ---> ;             : (20, 0)
Token ---> while         : (36, 0)
```



```
Token ---> 2             : (5, 2)
Token ---> ;             : (20, 0)
Token ---> while         : (36, 0)
Token ---> (             : (7, 0)
Token ---> j             : (4, j)
Token ---> <=            : (22, 0)
Token ---> k             : (4, k)
Token ---> )             : (8, 0)
Token ---> {             : (37, 0)
Token ---> rem           : (4, rem)
Token ---> =             : (23, 0)
Token ---> i             : (4, i)
Token ---> %             : (2, 0)
Token ---> j             : (4, j)
Token ---> ;             : (20, 0)
Token ---> if            : (32, 0)
Token ---> (             : (7, 0)
Token ---> rem           : (4, rem)
Token ---> ==            : (24, 0)
Token ---> 0             : (5, 0)
Token ---> )             : (8, 0)
Token ---> prime         : (4, prime)
Token ---> =             : (23, 0)
Token ---> 0             : (5, 0)
Token ---> ++            : (12, 0)
Token ---> j             : (4, j)
Token ---> ;             : (20, 0)
Token ---> }             : (39, 0)
Token ---> if            : (32, 0)
Token ---> (             : (7, 0)
Token ---> prime         : (4, prime)
Token ---> ==            : (24, 0)
Token ---> 1             : (5, 1)
Token ---> )             : (8, 0)
Token ---> write         : (4, write)
Token ---> (             : (7, 0)
Token ---> 1             : (5, 1)
Token ---> )             : (8, 0)
Token ---> ;             : (20, 0)
Token ---> ++            : (12, 0)
Token ---> i             : (4, i)
Token ---> ;             : (20, 0)
Token ---> }             : (39, 0)
Token ---> }             : (39, 0)
Token --->               : (29, 0)

C:\Users\2015305084\source\repos\1025\Debug>
```