[ Scanner.h ]

먼저 헤더 파일 내 구조체 타입을 살펴보면,

struct tokenType {

int number; // token number

int linenum;

int columnum;

union {

char id[ID\_LENGTH];

int num;

double fnum;

char str[MAX\_LENGTH];

char chr;

char comment[MAX\_LENGTH];

} value; // token value

};

기존의 토큰 타입 구조체에 확장할 키워드들(문자, 실수, 반복문 등)을 위해 linenum, columnum 변수를 생성하였고, 구조체 내 유니온에 fnum, str, chr, comment의 변수를 추가하였다.

또한 열거형 tsymbol에 기존의 39번에서, 확장할 키워드들의 목록을 53번까지 추가하였다.

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 41 \*/

twhile, tlbrace, tor, trbrace, tchar, tdouble,

/\* 42 43 44 45 46 47 \*/

tfor, tdo, tgoto, tswitch, tcase, tbreak,

/\* 48 49 50 51 52 53 \*/

tdefault, tcolon, tlchar, tlstring, tlcomment, tldouble

};

[ Scanner.cpp ]

**struct tokenType scanner(char file[])**

char fileName[30]; // file name

int linenum = 0; int columnum;

cpp 파일 내부에서는 우선적으로 전역 변수에 fileName, linenum, columnum 을 선언해주었고,

const 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 41 \*/

"while", "{", "||", "}", "char", "double",

/\* 42 43 44 45 46 47 \*/

"for", "do", "goto", "switch", "case", "break",

/\* 48 49 50 51 52 53 \*/

"default", ":", "%tlchar", "%tlstring", "%tlcomment", "%tldouble"

};

const char\* keyword[NO\_KEYWORD] = {

"const", "else", "if", "int", "return", "void", "while",

"char", "double", "for", "do", "goto", "switch", "case",

"break", "default"

};

enum tsymbol tnum[NO\_KEYWORD] = {

tconst, telse, tif, tint, treturn, tvoid, twhile,

tchar, tdouble, tfor, tdo, tgoto, tswitch, tcase,

tbreak, tdefault

};

tokenName, keyword, tnum에 각각 헤더파일에서 추가했던 열거형 키워드에 맞게 확장할 타입들을 추가했다.

scanner 함수를 살펴보면, 함수 시작시 전역 선언한 fileName에 파일의 이름을 복사하고, 기존과 달리 chr, srt, idx, isdouble 변수를 추가로 생성하며 함수를 시작한다.

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

이전과 달리 공백문자를 만날경우 바로 넘어가지 않고, linenum과 columnnum에 변화를 주어, 커서가 몇번째 줄의 몇번째 칸인지 저장해둔다.

double num = getNumber(ch, isdouble);

if (isdouble == true) {

token.number = tldouble;

token.value.fnum = num;

}

else {

token.number = tnumber;

token.value.num = (int)num;

}

숫자일 경우에, 정수인지 실수인지를 구분하고, double형인 경우 정수부와 실수부를 따로하여 저장한다.

case '\'': // char

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

token.number = tlchar;

token.linenum = linenum;

token.columnum = columnum;

token.value.chr = ch;

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

if (ch != '\'') lexicalError(5);

case '"': // string

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

token.number = tlstring;

token.linenum = linenum;

token.columnum = columnum;

idx = 0;

while (ch != '"') {

token.value.str[idx++] = ch;

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

ch = ' ';

}

else columnum++;

}

case '.': // 여기도 double

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

if (isdigit(ch)) {

int pointnum = 1;

double num = 0;

token.number = tnumber;

token.linenum = linenum;

token.columnum = columnum;

do {

num += (double)(ch - '0') \* (double)pow(0.1, pointnum++);

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

} while (isdigit(ch));

token.value.num = num;

}

특수 문자의 경우, 역슬래시, 큰따옴표, 마침표의 경우를 새롭게 추가해주고, 이외의 기존 항목에 대해서는 공백문자와 마찬가지로 linenum, columnum에 대한 변화를 지정해주었다.

**int getNumber(char firstCharacter, bool isdouble)**

먼저 기존의 함수 형태에서 매개변수 isdouble을 추가하여 숫자인지 판별시 정수인지 실수인지에 따라 다른 방법으로 숫자를 파싱하기 위해 매개변수를 추가하였다.

pointnum 변수를 추가로 선언하였고, 첫번째 문자가 0인경우에

else if (ch == '.') { // double

isdouble = true;

do {

num += (double)(ch - '0') \* (double)pow(0.1, pointnum++);

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

} while (isdigit(ch));

}

double형 변수일 때의 조건을 추가하여 소숫점 이하의 부분을 처리하는 과정을 만들었고, 그 이외의 경우에

if (ch == '.') { // double

isdouble = true;

do {

num += (double)(ch - '0') \* (double)pow(0.1, pointnum++);

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

} while (isdigit(ch));

}

else {

num = 10 \* num + (int)(ch - '0');

ch = fgetc(sourceFile);

if (ch == '\n') {

linenum++;

columnum = 1;

}

else columnum++;

}

}

double 형일때의 경우를 추가하여 같은 방법으로 처리하는 과정을 추가하였다. 추가적으로 이 함수또한 모든 경우에 있어서 linenum, columnum을 갱신해주었다.

**void printToken(struct tokenType token)**

if (token.number == tident)

printf("Token ---> %s (%d, %s, %s, %d, %d)\n", token.value.id, token.number, token.value.id, fileName, token.linenum, token.columnum);

else if (token.number == tnumber) {

printf("Token ---> %d (%d, %d, %s, %d, %d)\n", token.value.num, token.number, token.value.num, fileName, token.linenum, token.columnum);

}

else if (token.number == tldouble) {

printf("Token ---> %f (%d, %f , %s, %d, %d)\n", token.value.fnum, token.number, token.value.fnum, fileName, token.linenum, token.columnum);

}

else if (token.number == tlcomment)

printf("Documented Comments ---> %s \n", token.value.comment);

else {

printf("Token ---> %s (%d, %d, %s, %d, %d)\n", tokenName[token.number], token.number, 0, fileName, token.linenum, token.columnum);

}

모든 경우에 있어서의 출력 형식을 요구사항대로 변경하였고, double형일 때, 주석일 때의 출력을 추가해주었다.