REPORT

(컴파일러 중간고사)



|  |  |
| --- | --- |
| 제목 | 스캐너 |
| 제출일자 | 21.10.29 |
| 소속학과 | 컴퓨터공학과 |
| 학번 | 2017305039 |
| 성명 | 신동민 |

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

Token ---> const : (30, 0)

Token --->int : (33, 0)

Token ---> max : (4, max)

**프로그램 동작:** 1. 파일 입력 -> 2. 스캐너 -> 3. 토큰 인식 -> 4. 출력

Perfect.mc

**파일을 입력**

**프로그램**

**반복문**

인식이 가능한 토큰을

찾고 반환함

**Scanner()**

**토큰 출력**

토큰마다 형식에

맞는 출력

**<Main.cpp>**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

//키워드는 7개, 문자의 최대 길이는 12

#define NO\_KEYWORDS 7

#define ID\_LENGTH 12

//임시로 문자를 받을 공간

char id[ID\_LENGTH];

//인식할 수 있는 심벌들

enum tsymbol {

tnull = -1,

/\* 0 1 2 3 4 5 \*/

tnot, tnotequ, tmod, tmodAssign, tident, tnumber,

/\* 6 7 8 9 10 11 \*/

tand, tlparen, trparen, tmul, tmulAssign, tplus,

/\* 12 13 14 15 16 17 \*/

tinc, taddAssign, tcomma, tminus, tdec, tsubAssign,

/\* 18 19 20 21 22 23 \*/

tdiv, tdivAssign, tsemicolon, tless, tlesse, tassign,

/\* 24 25 26 27 28 29 \*/

tequal, tgreat, tgreate, tlbracket, trbracket, teof,

// 키워드 7개

/\* 30 31 32 33 34 35 36 \*/

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

/\* 37 38 39 \*/

tlbrace, tor, trbrace

};

//토큰에 대한 정보를 가진 구조체

struct tokenType {

int number; // token number

union {

char id[ID\_LENGTH];

int num;

} value; // token value

};

//이진탐색으로 찾을 키워드 7개

const char\* keyword[] = {

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

};

//이진탐색으로 찾은 키워드의 token number 배정할 때 사용

enum tsymbol tnum[] = {

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

};

//발생할 수 있는 에러 함수

void lexicalError(int n)

{

printf(" \*\*\* Lexical Error : ");

switch (n) {

//12 자를 넘김

case 1: printf("an identifier length must be less than 12.\n");

break;

//& 다음으로 &이 아닌 다른 문자가 올 때

case 2: printf("next character must be &.\n");

break;

//| 다음으로 |이 아닌 다른 문자가 올 때

case 3: printf("next character must be |.\n");

break;

//인식할 수 없음

case 4: printf("invalid character!!!\n");

break;

}

}

//첫글자가 문자인지 확인하는 함수

int superLetter(char ch)

{

if (isalpha(ch) || ch == '\_') return 1;

else return 0;

}

//글자가 문자나 숫자인지 확인하는 함수

int superLetterOrDigit(char ch)

{

if (isalnum(ch) || ch == '\_') return 1;

else return 0;

}

//문자로 읽은 숫자 값을 상수로 변환하는 함수

int getIntNum(char firstCharacter, FILE\* source\_file)

{

int num = 0;

char ch;

ch = firstCharacter;

//상수로 변환하는 반복문

do {

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

ch = fgetc(source\_file);

} while (isdigit(ch)); // 숫자가 아닐 때 까지 반복

ungetc(ch, source\_file); // retract

return num;

}

//파일을 받아 토큰을 찾는 함수

struct tokenType scanner(FILE\* source\_file)

{

struct tokenType token;

int i, j, k;

char ch;

token.number = tnull;

do {

//state 1: 공백제거

while (isspace(ch = fgetc(source\_file)));

i = 0;

//첫 글자가 문자면

if (superLetter(ch)) {

//글자를 임시로 저장

do {

if (i < ID\_LENGTH) id[i++] = ch;

ch = fgetc(source\_file);

} while (superLetterOrDigit(ch)); // 문자나 숫자가 아닐 때까지

if (i >= ID\_LENGTH) lexicalError(1);

id[i] = '\0';

ungetc(ch, source\_file); // retract

//이진탐색: find the identifier in the keyword table

i = 0;

j = NO\_KEYWORDS - 1;

do {

k = (i + j) / 2;

if (strcmp(id, keyword[k]) >= 0) i = k + 1;

if (strcmp(id, keyword[k]) <= 0) j = k - 1;

} while (i <= j);

// found, keyword exit

if ((i - 1) > j) { // state 4 : keyword

token.number = tnum[k];

strcpy(token.value.id, id);

}

// not found, identifier exit

else { // state 3 : identifier

token.number = tident;

strcpy(token.value.id, id);

}

}// end of identifier or keyword

//첫 글자가 숫자면

else if (isdigit(ch)) {

token.number = tnumber;

token.value.num = getIntNum(ch, source\_file);

}

//구분자, 연산자를 구분

else {

token.value.id[i++] = ch;

switch (ch) {

case '/': // state 10

ch = fgetc(source\_file);

if (ch == '\*') { // text comment

ch = fgetc(source\_file);

do {

while (ch != '\*') ch = fgetc(source\_file);

ch = fgetc(source\_file);

} while (ch != '/');

}

else if (ch == '/') // line comment

while (fgetc(source\_file) != '\n');

else if (ch == '=') { token.number = tdivAssign; token.value.id[i++] = ch; }

else {

token.number = tdiv;

ungetc(ch, source\_file); // retract

}

break;

case '!': // state 17

ch = fgetc(source\_file);

if (ch == '=') { token.number = tnotequ; token.value.id[i++] = ch; }

else {

token.number = tnot;

ungetc(ch, source\_file); // retract

}

break;

case '%': // state 20

ch = fgetc(source\_file);

if (ch == '=') { token.number = tmodAssign; token.value.id[i++] = ch; }

else {

token.number = tmod;

ungetc(ch, source\_file); // retract

}

break;

case '&': // state 23

ch = fgetc(source\_file);

if (ch == '&') { token.number = tand; token.value.id[i++] = ch; }

else {

lexicalError(2);

ungetc(ch, source\_file); // retract

}

break;

case '\*': // state 25

ch = fgetc(source\_file);

if (ch == '=') { token.number = tmulAssign; token.value.id[i++] = ch; }

else {

token.number = tmul;

ungetc(ch, source\_file); // retract

}

break;

case '+': // state 28

ch = fgetc(source\_file);

if (ch == '+') { token.number = tinc; token.value.id[i++] = ch; }

else if (ch == '=') { token.number = taddAssign; token.value.id[i++] = ch; }

else {

token.number = tplus;

ungetc(ch, source\_file); // retract

}

break;

case '-': // stats 32

ch = fgetc(source\_file);

if (ch == '-') { token.number = tdec; token.value.id[i++] = ch; }

else if (ch == '=') { token.number = tsubAssign; token.value.id[i++] = ch;}

else {

token.number = tminus;

ungetc(ch, source\_file); // retract

}

break;

case '<': // state 36

ch = fgetc(source\_file);

if (ch == '=') { token.number = tlesse; token.value.id[i++] = ch; }

else {

token.number = tless;

ungetc(ch, source\_file); // retract

}

break;

case '=': // state 39

ch = fgetc(source\_file);

if (ch == '=') { token.number = tequal; token.value.id[i++] = ch; }

else {

token.number = tassign;

ungetc(ch, source\_file); // retract

}

break;

case '>': // state 42

ch = fgetc(source\_file);

if (ch == '=') { token.number = tgreate; token.value.id[i++] = ch; }

else {

token.number = tgreat;

ungetc(ch, source\_file); // retract

}

break;

case '|': // state 45

ch = fgetc(source\_file);

if (ch == '|') { token.number = tor; token.value.id[i++] = ch; }

else {

lexicalError(3);

ungetc(ch, source\_file); // retract

}

break;

//구분자와 파일의 끝

case '(': token.number = tlparen; break;

case ')': token.number = trparen; break;

case ',': token.number = tcomma; break;

case ';': token.number = tsemicolon; break;

case '[': token.number = tlbracket; break;

case ']': token.number = trbracket; break;

case '{': token.number = tlbrace; break;

case '}': token.number = trbrace; break;

case EOF: token.number = teof; break;

//이곳에 없다면 인식할 수 없는 글자

default: {

printf("Current character : %c", ch);

lexicalError(4);

break;

}

} // end of switch

token.value.id[i] = '\0';

}// end of else

} while (token.number == tnull); //토큰을 찾을 때까지 반복

//찾은 토큰 반환

return token;

} // end of scanner

void main(int argc, char\* argv[])

{

FILE\* source\_file;

struct tokenType token;

//파일을 읽는데 없다면 종료

if ((source\_file = fopen("perfect.mc", "r")) == NULL) {

fprintf(stderr, "file not found \n");

exit(0);

}

do {

//파일내용을 스캐너로 토큰을 하나 찾음

token = scanner(source\_file);

//토큰이 숫자일때

if (token.number == tnumber) {

printf("Token ---> %-12d : (%3d, %12d)\n", token.value.num, token.number, token.value.num);

}

//토큰이 명칭일때

else if (token.number == tident) {

printf("Token ---> %-12s : (%3d, %12s)\n", token.value.id, token.number, token.value.id);

}

//토큰이 파일 끝일 때

else if (token.number == teof) {

break;

}

//토큰이 나머지의 경우

else {

printf("Token ---> %-12s : (%3d, %12d)\n", token.value.id, token.number, 0);

}

} while (1); // 파일이 끝날 때까지 반복

fclose(source\_file);

}

**<perfect.mc>**

const int max = 500;

void main() {

int i, j, k;

int rem, sum;

i = 2;

while (i <= max) {

sum = 0;

k = i / 2;

j = 1;

while (j <= max) {

sum = 0;

k = i / 2;

j = 1;

while (j <= k) {

rem = i % j;

if (rem == 0) sum += j;

++j;

}

if (i == sum)write(i);

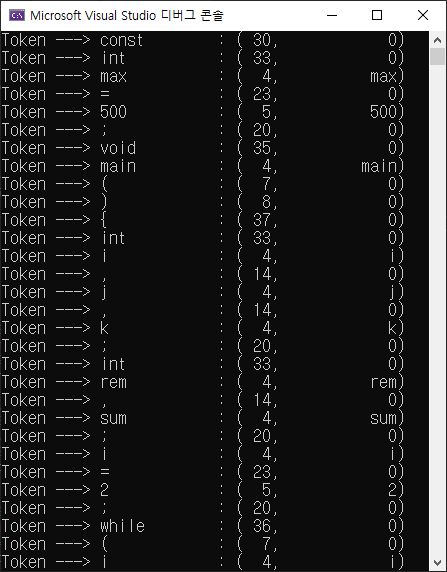
++i;

}

}

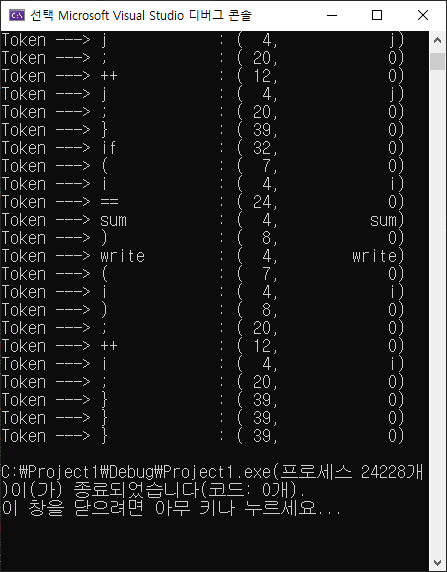
}

**결과 출력)**



●

●



설명: 프로그램의 출력결과의 처음과 끝을 캡처했습니다. 이런 결과가 나온 과정을 예시를 들어 설명하자면

Ex) const

c가 첫 글자로 오며 다음 글자들도 문자로 id[]변수에 임시로 저장되고 이진탐색을 통해서 해당 토큰이 키워드로 인식되어 키워드인 심벌로 등록된 값 30이 토큰 넘버가 되며 토큰 값은 0이 출력됩니다.

Ex) int

i가 첫 글자로 오며 다음 글자들도 문자로 id[]변수에 임시로 저장되고 이진탐색을 통해서 해당 토큰이 키워드로 인식되어 키워드인 심벌로 등록된 값 33이 토큰 넘버가 되며 토큰 값은 0이 출력됩니다.

Ex) max

m이 첫 글자로 오며 다음 글자들도 문자로 id[]변수에 임시로 저장되고 이진탐색을 통해서 해당 토큰이 키워드로 인식되지 않고 명칭으로 인식되어 명칭의 심벌로 등록된 값 4가 토큰 넘버가 되며 토큰 값은 max가 출력됩니다.

이렇게 반복문을 통해서 파일이 끝날 때까지 토큰을 인식해서 출력해주며 만약 인식할 수 없는 토큰이 생기면 에러문을 출력합니다.