컴퓨터 메카트로닉스 공학부 2018100904 권지현

**[소스코드]**

**prime.mc 🡺** **테스트용 샘플 프로그램(mini C 소스코드)**

const int max = 100;

void main()

{

int i, j, k;

int rem, prime;

i = 2;

while (i <= max)

{

prime = 1;

k = i / 2;

j = 2;

while (j <= k )

{

rem = i%j;

if(rem ==0) prime = 0;

++j;

}

if (prime == 1) write(i);

++i;

}

}

**scanner.cpp 🡺 어휘분석기 응용프로그램**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#define NO\_KEYWORDS 7

#define ID\_LENGTH 12

char id[ID\_LENGTH];

enum tsymbol {

tnull = -1,

tnot, tnotequ, tmod, tmodAssign, 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 \*/

twhile, tlbrace, tor, trbrace

};

struct tokenType {

int number;

union {

char id[ID\_LENGTH];

int num;

} value;

};

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

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

};

enum tsymbol tnum[NO\_KEYWORDS] = {

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

};

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 hexValue(char ch)

{

switch (ch) {

case '0': case '1': case '2': case '3': case '4':

case '5': case '6': case '7': case '8': case '9':

return (ch - '0');

case 'A': case 'B': case 'C': case 'D': case 'E': case 'F':

return (ch - 'A' + 10);

case 'a': case 'b': case 'c': case 'd': case 'e': case 'f':

return (ch - 'a' + 10);

default: return -1;

}

}

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

{

int num = 0;

int value;

char ch;

if (firstCharacter != '0') {

ch = firstCharacter;

do {

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

ch = fgetc(source\_file);

} while (isdigit(ch));

}

else {

ch = fgetc(source\_file);

if ((ch >= '0') && (ch <= '7'))

do {

num = 8 \* num + (int)(ch - '0');

ch = fgetc(source\_file);

} while ((ch >= '0') && (ch <= '7'));

else if ((ch == 'X') || (ch == 'x')) {

while ((value = hexValue(ch = fgetc(source\_file))) != -1)

num = 16 \* num + value;

}

else num = 0;

}

ungetc(ch, stdin);

return num;

}

struct tokenType scanner(FILE\* source\_file)

{

struct tokenType token;

int i, index;

char ch;

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';

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(token.value.id, id);

}

}

else if (isdigit(ch)) {

token.number = tnumber;

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

}

else switch (ch) {

case '/':

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 '!':

ch = fgetc(source\_file);

if (ch == '=') token.number = tnotequ;

else {

token.number = tnot;

ungetc(ch, stdin);

}

break;

case '%':

ch = fgetc(source\_file);

if (ch == '=')

token.number = tmodAssign;

else {

token.number = tmod;

ungetc(ch, stdin);

}

break;

case '&':

ch = fgetc(source\_file);

if (ch == '&') token.number = tand;

else {

lexicalError(2);

ungetc(ch, stdin);

}

break;

case '\*':

ch = fgetc(source\_file);

if (ch == '=') token.number = tmulAssign;

else {

token.number = tmul;

ungetc(ch, stdin);

}

break;

case '+':

ch = fgetc(source\_file);

if (ch == '+') token.number = tinc;

else if (ch == '=') token.number = taddAssign;

else {

token.number = tplus;

ungetc(ch, stdin);

}

break;

case '-':

ch = fgetc(source\_file);

if (ch == '-') token.number = tdec;

else if (ch == '=') token.number = tsubAssign;

else {

token.number = tminus;

ungetc(ch, stdin);

}

break;

case '<':

ch = fgetc(source\_file);

if (ch == '=') token.number = tlesse;

else {

token.number = tless;

ungetc(ch, stdin);

}

break;

case '=':

ch = fgetc(source\_file);

if (ch == '=') token.number = tequal;

else {

token.number = tassign;

ungetc(ch, stdin);

}

break;

case '>':

ch = fgetc(source\_file);

if (ch == '=') token.number = tgreate;

else {

token.number = tgreat;

ungetc(ch, stdin);

}

break;

case '|':

ch = fgetc(source\_file);

if (ch == '|') token.number = tor;

else {

lexicalError(3);

ungetc(ch, stdin);

}

break;

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;

default: {

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

lexicalError(4);

break;

}

} // switch end

} while (token.number == tnull);

return token;

} // end of scanner

int 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) {

fprintf(stdout, "%d \t (%d, %d)\n", token.value.num, token.number, token.value.num);

}

else if (token.number == 4) {

fprintf(stdout, "%s \t (%d, %s)\n",token.value.id, token.number, token.value.id);

}

else {

for (i = 0; i < ID\_LENGTH; i++)

fprintf(stdout, "%c", id[i]);

fprintf(stdout,"(%d, %d)\n", token.number, 0);

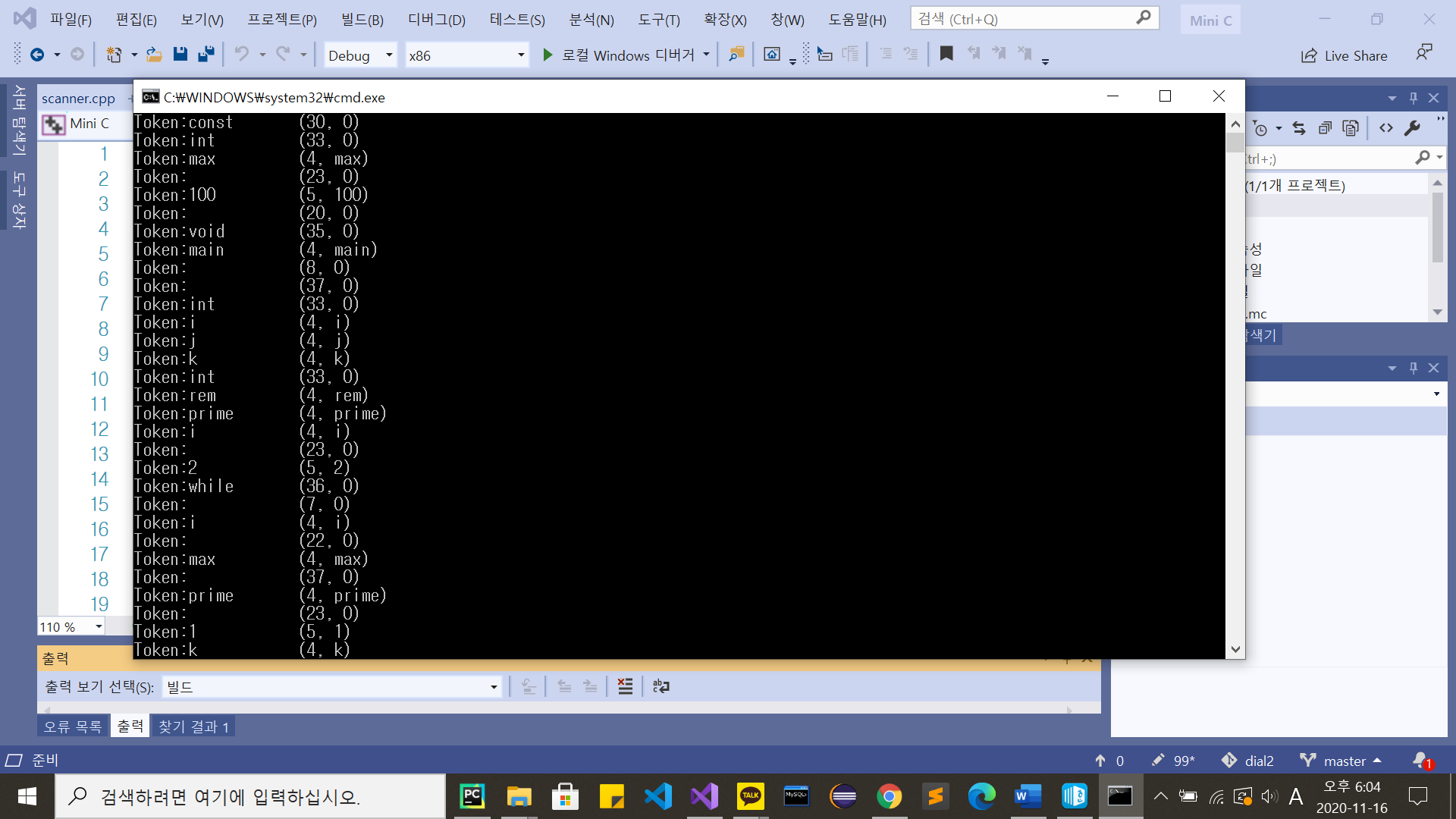
}

} while (!feof(source\_file));

fclose(source\_file);

}

**[ 출력 결과 ] 위의 prime.mc의 파일을 읽어 어휘분석기에서 돌린 결과 화면 출력**



**아래 부분 출력결과는 prime.mc**

**파일 안의 소스코드 전체를 분석한**

**결과**

**const int max = 100 ;**

**\* = ; < 와 같은 기호가 심벌테이블의 인덱스로는 출력이 되지만 스트링 값으로는 출력이 안되는 문제점이 발생하였습니다. 기호 외에는 모두 정상적으로 출력되었습니다.**

**[작성 소감]**

4단원 어휘분석기 학습을 마친 후 어휘분석기를 직접 구현해보면서 그 동안 배웠던 스캐너에 대한 이해도가 더 높아졌습니다. 스캐너의 역할인 ‘소스 프로그램을 읽어 들여 일련의 토큰을 생성하는 일’을 하는 것을 직접 결과 화면을 통해 접해보니 정확히 그 기능을 알게 되기도 하였습니다. 하지만 직접 구현하다 보니 하는 과정에서 몇 가지 어려움이 있었습니다.

가장 근본적으로 소스코드 작성에 대한 어려움 이였습니다. 물론 많은 부분을 책을 참고하면서 따라 치는 형식으로 과제를 수행하였지만 어떠한 방법으로 코드를 구성해야 하는지에 대한 고민을 하게 되었습니다. 따라서 교수님께서 올리신 어휘분석기 구현방법과 책을 함께 보면서 여러 차례 시행착오 결과 구성하는 방법에 대한 해답을 찾게 되었습니다.

두 번째로는 prime.mc 소스프로그램을 어휘분석기에 어떤 방법으로 읽히도록 해야 하는지에 대한 고민이 있었습니다. 이 방법은 구글링을 통한 검색을 통해 알게 되었습니다. 결과 디버깅을 클릭 후 프로젝트 속성의 디버깅 옵션에 명령 인수를 prime.mc로 설정해준 후 scanner.cpp와 같은 프로젝트에 해당 소스프로그램을 따로 작성하여 한 프로젝트로 구성해주었습니다. 그 이후 어휘분석기에서도 해당 소스프로그램을 읽어 들일 수 있도록 그에 맞게 코드를 작성하였습니다.

마지막으로 전체 코드 작성 후 결과확인에 있어 한가지 문제점이 발견되었습니다. =. ;, <, > 등 과 같은 기호가 심벌테이블의 인덱스로는 출력이 되지만 스트링 값으로는 출력이 안되는 문제점이 발생하였습니다. 기호 외에는 모두 정상적으로 출력되었습니다. 위와 같은 오류사항을 해결하기 위해 작성한 소스코드를 책과 파일과 대조해 계속 비교해봤지만 어디에서 잘못 작성했는지 발견하지 못했습니다. 디버그 시 오류도 나오지 않아 더욱 발견하기에 어려움이 있었던 것 같습니다.

과제가 나온 순간부터 작성하고 계속 오류사항을 확인해봤지만 결국 과제 제출을 하루 앞두고 이를 발견하지 못해 해결하지 못하고 과제작성을 하게 되었습니다. 과제에 대한 완성도가 약간 떨어졌지만 직접 어휘분석기를 구현하는 과정에서 분명 이해도가 엄청 높아졌음을 확신할 수 있습니다. 또한 여러 차례 어려움을 겪으면서 오류 해결에 대한 자신감도 붙게 되었습니다.