

Homework #01

- CH4. Lexical Analysis -

과목명	프로그래밍 언어
교수명	최종선 교수님
학 과	IT대학 컴퓨터학부
학 번	20172655
이 름	이강산
제출일	2021.05.14. 금요일

< 목차 >

1. 구현 개요

- lexical analyzer란
- 구현 기능(C, python)

2. 소스 코드

- C언어로 작성한 코드
- python으로 재작성한 코드 (C언어와의 차이)

3. 실행 결과

- 입력파일
- 실행결과(C)
- 실행결과(python)

4. 구현을 통해 배운 점

1. 구현 개요

- Lexical Analyzer

컴파일러를 구성하는 첫 번째 단계인 Lexical Analyzer는 주어진 source code를 scanning 하면서 정규 문법에 따라 의미가 있는 단위로 문자열을 묶는다. 이 과정에서 statement들은 token 단위로 쪼개지며, token들은 컴파일러 내부에서 효율적으로 다루기 쉬운 정수값으로 표현된다.

Lexical Analyzer를 C와 python으로 구현하며 컴파일러 동작의 첫 번째 단계인 구문 분석에 대해 연구한다. 구문 분석에 사용되는 입력 소스코드(front01.c, front02.py)는 명세에 주어진 구문 분석 기능 내에서 작성되었다. chapter4에서 주어진 lexical analyzer를 바탕으로 구현한 기능들은 다음과 같다.

- 구현기능(C)

1) 주어진 기능에 추가

- getChar() : 입력으로부터 다음번째 문자를 가져와 그 문자 유형을 결정하는 함수. 변수 명의 _(언더바), 숫자의 소숫점 인식하도록 기능 추가.
- lookup(char) : nextChar을 lexeme에 추가하는 함수.
‘+’ 인식 시 증감 연산인지 확인하는 기능 추가
- lex() : 산술식을 위한 단순 어휘분석기, int-float, increment(++) 연산 판단 코드 추가

2) 명세로 주어진 예약어 인식

```
for (FOR_CODE, 30) if (IF_CODE, 31) else (ELSE_CODE, 32)
while (WHILE_CODE, 33) do (DO_CODE, 34) switch (SWITCH_CODE, 37)
int (INT_CODE, 35) float (FLOAT_CODE, 36) char (CHAR_CODE 38)
```

3) 명세로 주어진 예약어들과 같이 사용되는 예약어 인식

```
case (CASE_CODE, 39) break (BREAK_CODE, 40)
continue (CONTINUE_CODE, 41) default (DEFAULT_CODE, 42)
```

4) 명세로 주어진 예약어들이 주로 사용하는 연산자, 기호 인식

```
% (MOD_OP, 63) ++ (INCRE_OP, 64) < (CMPRIGHT_OP, 65)
> (CMPLEFT_OP, 66) { (LEFT_BRACE, 60) } (RIGHT_BRACE, 61)
' (QUOTES, 67) : (COLON, 68) ; (SEMICOLON, 69)
```

5) 주어진 identifier가 예약어인지 확인

- reserved word lookup table
- lookup_ident(char) 함수 : identifier가 reserved word인지 판단

- 구현기능(python)

python의 경우 switch-case, do-while 같은 구문은 존재하지 않으며, 그 대신 elif, in 등 C언어에는 포함되지 않는 예약어들이 존재한다. 또한 ++같은 increment 연산이 없으며, 중괄호로 코드의 범위를 나타내는 대신 들여쓰기로 대신하며, statement의 끝을 알리는 세미콜론 또한 존재하지 않는다.

변수명과 기본적인 동작은 C언어로 작성된 프로그램과 동일하게 구현되었으나, 문법의 차이로 일부 차이점이 발생하였다.

1) 구현된 예약어

```
for (FOR_CODE, 30) in (IN_CODE, 31) if (IF_CODE, 32) elif (ELIF_CODE, 33)
else (ELSE_CODE, 34) while (WHILE_CODE, 35) break (BREAK_CODE, 36)
continue (DEFAULT_CODE, 37) return (RETURN_CODE, 38)
```

2) 구현된 예약어들이 주로 사용하는 연산자, 기호

```
= (ASSIGN_OP, 20) + (ADD_OP, 21) - (SUB_OP, 22)
* (MULT_OP, 23) / (DIV_OP, 24) % (MOD_OP, 63)
< (CMPRIGHT_OP, 65) > (CMPLEFT_OP, 66) ( (LEFT_PAREN, 25)
) (RIGHT_PAREN, 26) ' (QUOTES, 67) : (COLON, 68)
```

2. 소스 코드

- C언어로 작성한 코드 (주요 변경 사항)

1. reserved word lookup table / lookup_ident() 함수

```
#define RESERVED_WORD_CNT 13
char* reserved_word[13] = { "for", "if", "else", "while", "do", "int", "float", "switch",
"char", "case", "break", "continue", "default" };
/*****/
int lookup_ident() {
    int i;
    for (i = 0; i < RESERVED_WORD_CNT; i++)
    {
        if (strcmp(lexeme, reserved_word[i]) == 0)
        {
            nextToken = i + 30;
//예약어의 TOKEN CODE와 reserved_word_lookup_table의 index 가 30 차이나도록 설계
            return nextToken
        }
    }
    nextToken = IDENT;
    return nextToken;
}
```

identifier가 reserved word인지 파악하기 위해 사용되는 char* 타입 배열로, string literal들을 가지며 전역으로 선언되었다. main()의 while loop속에서 lex()를 수행하며 입력된 identifier를 lexeme 배열에 저장한 후, lexeme에 대해 lookup_ident()를 수행한다. 모든 reserved word에 대해 문자열 비교 수행 후, 일치하는 경우 존재 시 해당 값으로 token 변경, 일치하는 경우 존재하지 않으면 token은 identifier 그대로 유지된다.

2. increment 연산 구현 - lookup(ch) 함수, lex()함수

```
int incre_check = 0;
//set incre_check if current lexeme is increment operation
/*****/
int lookup(char ch) {
    ...
    case '+':
        addChar();
        if (incre_check == 1) nextToken = INCRE_OP;
        else nextToken = ADD_OP;
        break;
    ...
}
```

```

/*****/
int lex() {
...
    case UNKNOWN:
        lookup(nextChar);
        getChar();
        if (nextChar == '+')
        {
            incre_check = 1;
            lookup(nextChar);
            getChar();
        }
        else;
        break;

```

case UNKNOWN 수행 후 getChar()로부터 입력 받은 문자가 '+'일 때 increment 연산으로 판단한다. incre_check 변수를 사용함으로써 두 번째 lookup(ch)호출 시 token을 ADD_OP이 아닌 INCRE_OP으로 설정할 수 있게 된다.

3. identifier명 인식 확장, float 인식 - getChar(), lex()함수

```

void getChar() {
    if ((nextChar = getc(in_fp)) != EOF) { //읽어온 게 EOF 아니면 수행
        if (isalpha(nextChar) || nextChar == '_')
            //A-Z:1, a-z:2, 알파벳 아닐 시 0, 변수명에 _(언더바)포함 고려
            charClass = LETTER;
        else if (isdigit(nextChar) || nextChar == '.')
            //읽어온 문자가 숫자로 변경 가능하면 1, 아니면 0,
            //(소수점)은 float 판별 위해 추가
            {
                if (nextChar == '.') float_check = 1;
                charClass = DIGIT;
            }
    }
...
/*****/
int lex() {
...
    case LETTER:
        addChar();
        getChar();
        while (charClass == LETTER || charClass == DIGIT) {
            addChar();

```

```

        getChar();
    }
    lookup_ident();
    break;
case DIGIT:
    addChar();
    getChar();
    while (charClass == DIGIT) {
        addChar();
        getChar();
    }
    if(float_check==1)    nextToken = FLOAT_LIT;
    else nextToken = INT_LIT;
    break;
...

```

charClass가 LETTER, DIGIT인 경우 각각 getChar() 시 입력받을 수 있는 문자에 '_'와 '.'을 추가하여 '_'문자가 들어간 identifier명과 소숫점이 포함된 숫자를 lexeme으로 인식할 수 있게 한다. 소수의 경우 float_check변수를 두어 소숫점 입력 여부를 판단하여 lex()의 while loop 종료 후 FLOAT_LIT과 INT_LIT중 알맞은 값을 token에 배정한다.

- python으로 재작성한 코드 (C언어와의 차이)

1. 전역변수 선언, reserved word lookup table

```

global fp
global charClass
global lexeme
global nextChar
global nextToken
#####
reserved_word = ['for', 'in', 'if', 'elif', 'else', 'while', 'break', 'continue', 'return']
RESERVED_WORD_CNT = 9
#####
ASSIGN_OP = 20 #=
ADD_OP = 21 #+
SUB_OP = 22 #-

```

C언어로 작성된 코드와 마찬가지로 reserved_word를 담는 list, character class값, token 값은 전역 변수로 선언하였다. 구현된 각 함수에서 참조 및 값의 변경이 이루어지는 변수들은 global을 붙여 선언하였다.

2. if-elif-else문으로 switch-case 대체

```

def lookup(ch):
    global nextToken
    if ch=='=':
        addChar()
        nextToken = ASSIGN_OP
    elif ch=='+' :
        addChar()
        nextToken = ADD_OP
    elif ch=='-' :
        addChar()
        nextToken = SUB_OP
    elif ch=='*':
        addChar()
        nextToken = MULT_OP
    elif ch=='/':
        ...

```

python은 switch-case문에 대해 정의되지 않아 C로 작성된 프로그램과 동일한 동작을 보장하기 위해 if-elif-else문으로 대체하였다.

3. lexeme list 데이터 처리 방식

```

def addChar():
    global lexeme
    global nextChar
    lexeme.append(nextChar)
    return

#####
def lex():
    ...
    str="".join(lexeme)
    print("Next token is {0}, Next lexeme is {1}".format(nextToken, str))
    float_check = 0
    lexeme=[]
    ...

```

C언어에서 lexeme배열을 index를 통해 제어했던 것과 달리, python에서는 list의 append()함수를 통해 매 addChar() 호출마다 nextChar을 lexeme list에 추가한다. lexeme list는 lex()의 수행 완료 후 return 직전에 초기화된다.

3. 실행 결과

- 입력파일

front01.c	front02.py
<pre>int k = 0; float fv = 3.0; for (int i = 0; i < 10; i++) { if (i > 9) continue; k = k % 3; switch (k) { case 0: k = k + 1; break; case 1: k++; break; default: fv = fv - 0.1; break; } }</pre>	<pre>name='leekangsan' age=24 height=170.3 i=0 while(i<20) i=i+1 if(i%3==0) age=age+1 elif(i%3==1) age=age+2 else continue if i>15 break</pre>

front01.c : C언어의 입력파일은 int와 float, for문, increment 연산, switch-case문을 인식하는 지 판단할 수 있도록 작성되었다.

front02.py : python의 입력파일은 identifier들의 배정, while문, if-elif-else문을 인식하는 지 판단할 수 있도록 작성되었다. python의 변수들은 정적으로 타입을 선언하지 않으면 배정된 값에 따라 정해진다. 해당 입력파일의 name, age 등의 변수들은 프로그램 수행 시 identifier로 명시되도록 작성하였다.

- 실행결과(C)

C:\Users\San\Desktop\PLwc_20172655.exe

```
Next token is: 35, Next lexeme is int
Next token is: 11, Next lexeme is k
Next token is: 20, Next lexeme is =
Next token is: 10, Next lexeme is 0
Next token is: 69, Next lexeme is ;
Next token is: 36, Next lexeme is float
Next token is: 11, Next lexeme is fv
Next token is: 20, Next lexeme is =
Next token is: 62, Next lexeme is 3.0
Next token is: 69, Next lexeme is ;
Next token is: 30, Next lexeme is for
Next token is: 25, Next lexeme is (
Next token is: 35, Next lexeme is int
Next token is: 11, Next lexeme is i
Next token is: 20, Next lexeme is =
Next token is: 10, Next lexeme is 0
Next token is: 69, Next lexeme is ;
Next token is: 11, Next lexeme is i
Next token is: 65, Next lexeme is <
Next token is: 10, Next lexeme is 10
Next token is: 69, Next lexeme is ;
Next token is: 11, Next lexeme is i
Next token is: 64, Next lexeme is ++
Next token is: 26, Next lexeme is )
Next token is: 60, Next lexeme is {
Next token is: 31, Next lexeme is if
Next token is: 25, Next lexeme is (
Next token is: 11, Next lexeme is i
Next token is: 66, Next lexeme is >
Next token is: 10, Next lexeme is 9
Next token is: 26, Next lexeme is )
Next token is: 41, Next lexeme is continue
Next token is: 69, Next lexeme is ;
Next token is: 11, Next lexeme is k
Next token is: 20, Next lexeme is =
Next token is: 11, Next lexeme is k
Next token is: 63, Next lexeme is %
Next token is: 10, Next lexeme is 3
Next token is: 69, Next lexeme is ;
Next token is: 37, Next lexeme is switch
Next token is: 25, Next lexeme is (
Next token is: 11, Next lexeme is k
Next token is: 26, Next lexeme is )
Next token is: 60, Next lexeme is {
Next token is: 39, Next lexeme is case
Next token is: 10, Next lexeme is 0
Next token is: 68, Next lexeme is :
Next token is: 11, Next lexeme is k
Next token is: 20, Next lexeme is =
Next token is: 11, Next lexeme is k
Next token is: 64, Next lexeme is +
Next token is: 10, Next lexeme is 1
Next token is: 69, Next lexeme is ;
Next token is: 40, Next lexeme is break
Next token is: 69, Next lexeme is ;
Next token is: 39, Next lexeme is case
Next token is: 10, Next lexeme is 1
Next token is: 68, Next lexeme is :
Next token is: 11, Next lexeme is k
Next token is: 64, Next lexeme is ++
Next token is: 69, Next lexeme is ;
Next token is: 40, Next lexeme is break
Next token is: 69, Next lexeme is ;
Next token is: 42, Next lexeme is default
Next token is: 68, Next lexeme is :
Next token is: 11, Next lexeme is fv
Next token is: 20, Next lexeme is =
Next token is: 11, Next lexeme is fv
Next token is: 22, Next lexeme is -
Next token is: 62, Next lexeme is 0.1
Next token is: 69, Next lexeme is ;
Next token is: 40, Next lexeme is break
Next token is: 69, Next lexeme is ;
Next token is: 61, Next lexeme is }
Next token is: -1, Next lexeme is EOF
계속하려면 아무 키나 누르십시오 . . .
```

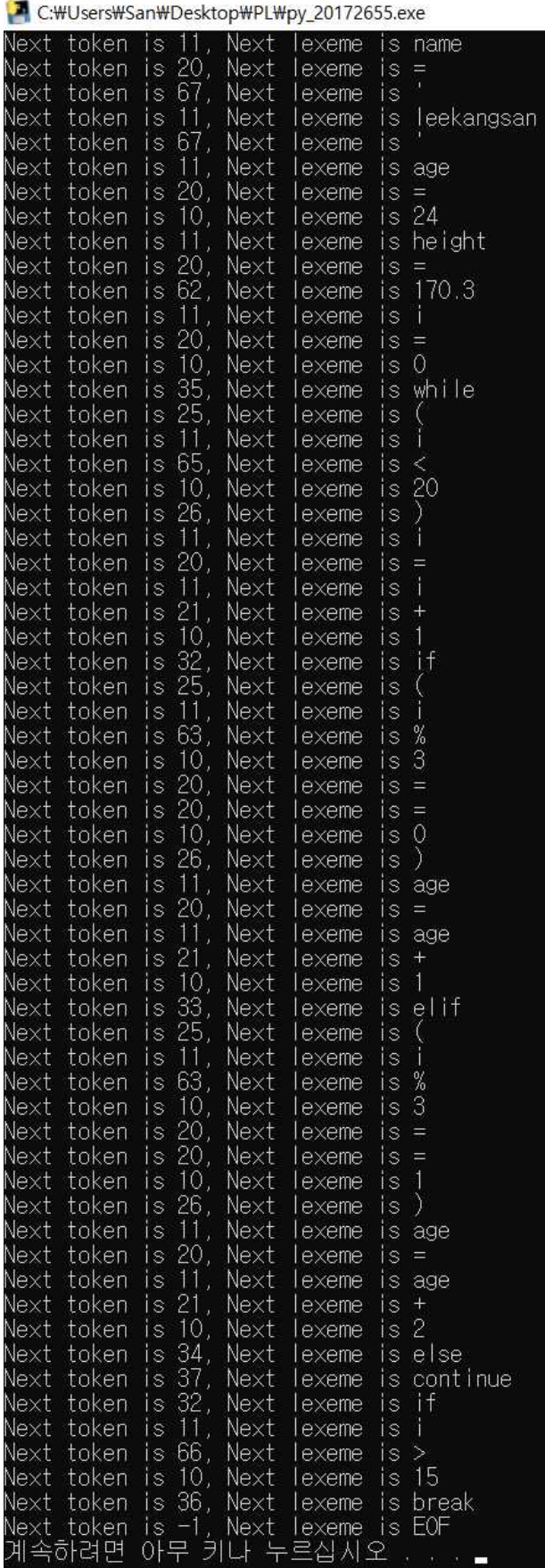
c_20172655.exe 실행 결과

다음과 비교해보면 올바르게 결과가
도출되었다는 것을 알 수 있다.

```
for (FOR_CODE, 30)
if (IF_CODE, 31)
else (ELSE_CODE, 32)
while (WHILE_CODE, 33)
do (DO_CODE, 34)
switch (SWITCH_CODE, 37)
int (INT_CODE, 35)
float (FLOAT_CODE, 36)
char (CHAR_CODE 38)
case (CASE_CODE, 39)
break (BREAK_CODE, 40)
continue (CONTINUE_CODE, 41)
default (DEFAULT_CODE, 42)
```

```
% (MOD_OP, 63)
++ (INCRE_OP, 64)
< (CMPRIGHT_OP, 65)
> (CMPLEFT_OP, 66)
{ (LEFT_BRACE, 60)
} (RIGHT_BRACE, 61)
' (QUOTES, 67)
: (COLON, 68)
; (SEMICOLON, 69)
```

- 실행결과(python)

 <p>C:\Users\San\Desktop\PLW\py_20172655.exe</p> <pre> Next token is 11, Next lexeme is name Next token is 20, Next lexeme is = Next token is 67, Next lexeme is ' Next token is 11, Next lexeme is leekangsan Next token is 67, Next lexeme is ' Next token is 11, Next lexeme is age Next token is 20, Next lexeme is = Next token is 10, Next lexeme is 24 Next token is 11, Next lexeme is height Next token is 20, Next lexeme is = Next token is 62, Next lexeme is 170.3 Next token is 11, Next lexeme is i Next token is 20, Next lexeme is = Next token is 10, Next lexeme is 0 Next token is 35, Next lexeme is while Next token is 25, Next lexeme is (Next token is 11, Next lexeme is i Next token is 65, Next lexeme is < Next token is 10, Next lexeme is 20 Next token is 26, Next lexeme is) Next token is 11, Next lexeme is i Next token is 20, Next lexeme is = Next token is 11, Next lexeme is i Next token is 21, Next lexeme is + Next token is 10, Next lexeme is 1 Next token is 32, Next lexeme is if Next token is 25, Next lexeme is (Next token is 11, Next lexeme is i Next token is 63, Next lexeme is % Next token is 10, Next lexeme is 3 Next token is 20, Next lexeme is = Next token is 20, Next lexeme is = Next token is 10, Next lexeme is 0 Next token is 26, Next lexeme is) Next token is 11, Next lexeme is age Next token is 20, Next lexeme is = Next token is 11, Next lexeme is age Next token is 21, Next lexeme is + Next token is 10, Next lexeme is 1 Next token is 33, Next lexeme is elif Next token is 25, Next lexeme is (Next token is 11, Next lexeme is i Next token is 63, Next lexeme is % Next token is 10, Next lexeme is 3 Next token is 20, Next lexeme is = Next token is 20, Next lexeme is = Next token is 10, Next lexeme is 1 Next token is 26, Next lexeme is) Next token is 11, Next lexeme is age Next token is 20, Next lexeme is = Next token is 11, Next lexeme is age Next token is 21, Next lexeme is + Next token is 10, Next lexeme is 2 Next token is 34, Next lexeme is else Next token is 37, Next lexeme is continue Next token is 32, Next lexeme is if Next token is 11, Next lexeme is i Next token is 66, Next lexeme is > Next token is 10, Next lexeme is 15 Next token is 36, Next lexeme is break Next token is -1, Next lexeme is EOF 계속하려면 아무 키나 누르십시오 . . . </pre>	<p>py_20172655.exe 실행결과</p> <p>다음과 비교해보면 올바르게 결과가 도출되었다는 것을 알 수 있다.</p> <pre> for (FOR_CODE, 30) in (IN_CODE, 31) if (IF_CODE, 32) elif (ELIF_CODE, 33) else (ELSE_CODE, 34) while (WHILE_CODE, 35) break (BREAK_CODE, 36) continue (DEFAULT_CODE, 37) return (RETURN_CODE, 38) = (ASSIGN_OP, 20) + (ADD_OP, 21) - (SUB_OP, 22) * (MULT_OP, 23) / (DIV_OP, 24) % (MOD_OP, 63) < (CMPRIGHT_OP, 65) > (CMPLEFT_OP, 66) ((LEFT_PAREN, 25)) (RIGHT_PAREN, 26) ' (QUOTES, 67) : (COLON, 68) </pre>
---	--

4. 구현을 통해 배운 점

컴파일러 동작의 첫 번째 단계인 어휘 분석에서 컴파일러는 입력받은 소스코드를 스캐닝하여 의미있는 단어들을 검출해낸다. 어휘 분석은 구문 분석을 위한 front-end 이며, 일련의 과정을 처리하기 위해 어휘 분석기를 사용한다.

어휘분석기(Scanner)는 문자열 패턴 매칭기로서 동작하며, 사전에 주어진 문자열패턴을 토대로 소스코드로부터 substring을 검출한다. 이 때 검출되는 의미 있는 substring들을 모아 lexeme이라고 하며, 구조에 따라서 정수인 토큰 값을 배정한다. 이는 곧 키워드, 식별자, 연산자, 수치상수, 문자상수, 특수문자 등으로 분류된다.

소스코드는 토큰단위로 다음 단계인 Parser에게 전달되며 구문 분석에 사용된다. 토큰들은 구문분석기에 전달되어 추상 구문 트리(AST)가 되고, 이 과정에서 문법에 맞지 않는 표현을 파악한다. 이후 최적화와 코드 생성등의 과정을 거쳐 machine code가 생성된다.