|  |
| --- |
|  |
| [컴파일러 보고서] |
|  |
| [homework2] |

**[국민대학교 컴퓨터 공학부]**

2017 November 14

저자:20153208 이소영

[컴파일러 보고서]

[homework2]

아래 2개의 문법에 대하여 Recursive descent 파서와 Predictive parser를 각각 작성하시오.

1. Identifier list를 인식하는 문법

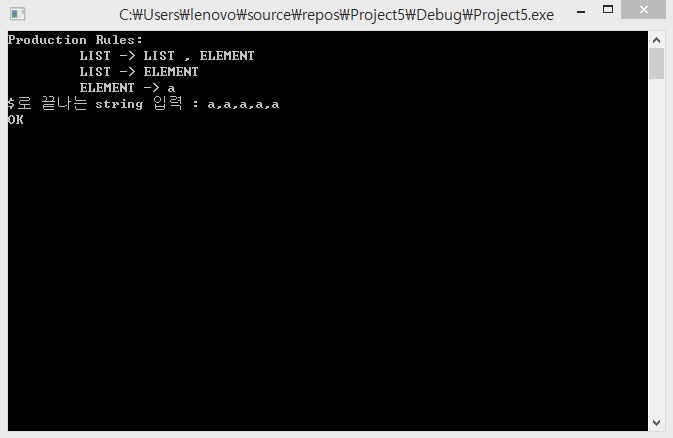
LIST -> LIST , ELEMENT

LIST -> ELEMENT

ELEMENT -> a

입력 예: "a,a,a,a,a"

Recursive descent parser



LIST -> LIST , ELEMENT

LIST -> ELEMENT

ELEMENT -> a

우순환으로 변경해주면,

LIST -> ELEMENT LIST'

LIST' -> , ELEMENT LIST'

LIST' -> ℇ

ELEMENT -> a

이다.

따라서 생성규칙은

A -> BC

B -> a

C -> bBC

C -> ℇ

가 된다.

따라서 A -> BC는

void pA()로 void pB(),pC();를 호출하고

Ch가 a면 pB()와 pC(); 를 호출하고 아닐 경우 에러를 출력한다.

B -> a

void pB()로 ch가 a이면 pa()를 호출하고 아닐 경우 에러를 출력한다.

C -> bBC

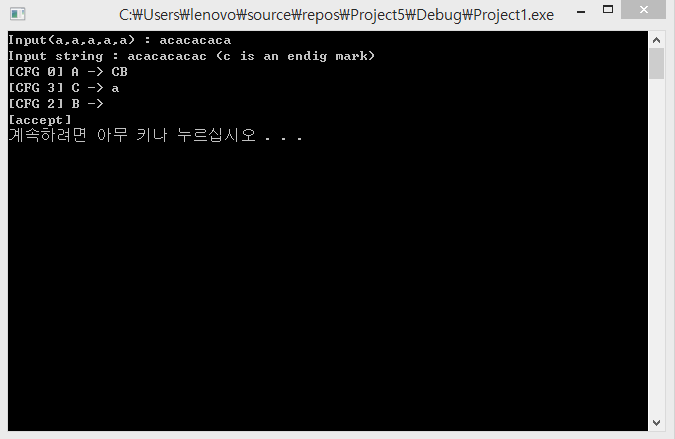
C -> ℇ

는 void pC()로 pb(),pB(), pC()를 호출하고

Ch가 ‘,’일 경우 pb(),pB(),pC()를 호출하고 ‘$’이거나 ‘\n’이면 break, 아닐 경우 에러를 호출한다.

main에서 ch에 getchar()를 해서 값을 얻고, pA()를 호출하여 판별한다.

Predictive parser



A -> CB

B -> bCB

B -> ℇ

C -> bBC

C -> ℇ

를 파싱테이블로 만들면,

a b $

------------------

A 0

B 1 2

C 3

으로 빈칸은 전부 -1로 채운다.

이 파싱테이블을 가지고 실습자료실에 실습10에 올라와있는 predivtive3.c

에서 load create 와 load parsing table를 위와 맞게 수정한다.

Load create을

0. A -> CB

1. B -> bCB

2. B -> epsilon

3. C -> a

에 따라

strcpy(create\_rule[0], "CB");

strcpy(create\_rule[1], "bCB");

strcpy(create\_rule[2], "");

strcpy(create\_rule[3], "a");

로 바꾼다.

Load\_parsing\_table

은 채워넣을 숫자만을 뽑아 위치에 집어넣어준다.

parsing\_table[0][0] = 0;

parsing\_table[1][1] = 1;

parsing\_table[1][2] = 2;

parsing\_table[2][0] = 3;

또한, 나머지는 -1로 다 채워넣어준다.

이를 위해 init\_table()를 만들어

for문으로 parsing\_table의 크기에 맞춰(논터미널과 터미널로) -1로 초기화시켜준다.

그 후 main에서 init\_table를 load\_create위에(초기화이기 때문에) 호출해준다.

2) 덧셈, 곱셈 수식을 인식하는 문법

E -> E + T

E -> T

T -> T \* F

T -> F

F -> ( E )

F -> a

    입력 예: "a+(a+a\*(a+a))\*a+(a)"

E -> E + T | T

T -> T \* F | F

F -> ( E ) | a

를 우순환으로 바꾸면

E -> TE'

E' -> +TE' | ℇ

T -> FT'

T' -> \*FT' | ℇ

F -> (E) | a

이다.

이것은

E -> TEP

EP -> +TEP

T -> FTP

TP -> \*FTP

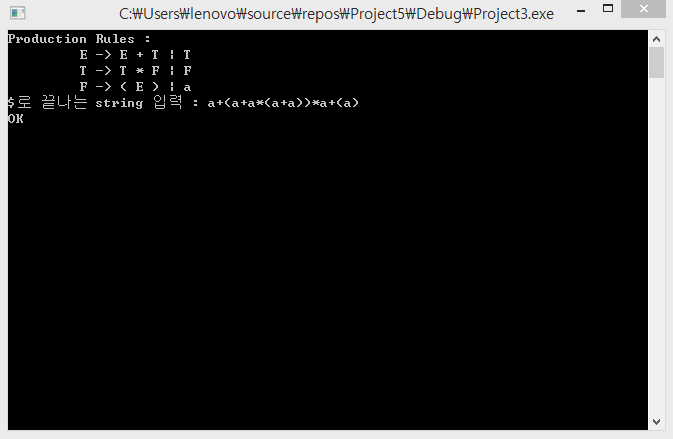
TP -> ℇ

F -> (E)

F -> a

가 된다.

Recursive descent parser



E -> TEP

은 void pE()를 void pT(),pEP()를 호출하고 ch가 ‘(‘이거나 ‘a’이면

pT(),pEP()를 호출한다. 아닐경우 에러를 호출한다.

EP -> +TEP

는 void pEP()를 void pplus(),pT(),pEP()를 호출하고 ch가 +인 경우 pplus(),pT(),pEP()를 호출한다. 아닐 경우 ch가 ‘)’이거나 ‘$’ 이거나 ‘\n’이면 break를 하고 둘 다 아닐경우 error를 호출한다.

T -> FTP

는 void pT()를 void pF(),pTP()를 호출하고 ch가 ‘(‘나 ‘a’일 경우 pF(),pTP()를 호출하고 아닐 경우 error를 호출한다.

TP -> \*FTP

TP -> ℇ

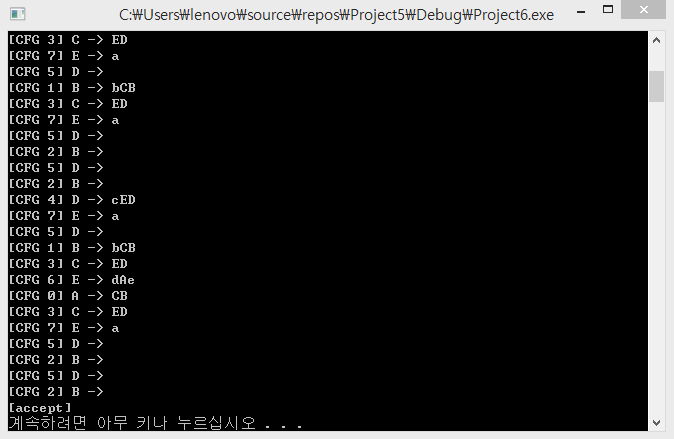
는 void pTP()를 void pmult(),pF(),pTP()를 호출하고 ch가 ‘\*’일 경우 pmult,pF(),pTP()를 호출하고 아니면 ch가 ‘+’거나 ‘$’이거나 ‘)’이거나 ‘\n’이면 break;하고 둘 다 아닐 경우 error를 호출한다.

F -> (E)

F -> a

는 void pF()를 void pleft(),pE(),pright(),pa()를 호출하고 ch가 ‘(‘이면 pleft(),pE(),pright()를 호출하고 아닐 경우 ch가 ‘a’면 pa()를 호출하고 둘다 아닐 경우 error를 호출한다.

Predictive parser



E -> TEP

EP -> +TEP

T -> FTP

TP -> \*FTP

TP -> ℇ

F -> (E)

F -> a

를 바꾸면

0.A -> CB

1.B -> +CB

2.B -> ℇ

3.C -> ED

4.D -> \*ED

5.D -> ℇ

6.E -> (A)

7.E -> a

로 바꾼다. 이것을 파싱 테이블로 만들면

Vn/Vt | a ( ) \* + $(d)

----------------------------------------

A | 0 0

B | 2 1 2

C | 3 3

D | 5 4 5 5

E | 6 7

로 하고 (는 b로 )는 c로 \*은 d로 +는 e 로 한다.

생성규칙에 따라 Load\_create은

strcpy(create\_rule[0], "CB");

strcpy(create\_rule[1], "eCB");

strcpy(create\_rule[2], "");

strcpy(create\_rule[3], "ED");

strcpy(create\_rule[4], "dED");

strcpy(create\_rule[5], "");

strcpy(create\_rule[6], "a");

strcpy(create\_rule[7], "bAc");

로 하고

Load parsing able은

parsing\_table[0][0] = 0;

parsing\_table[0][1] = 0;

parsing\_table[1][2] = 2;

parsing\_table[1][4] = 1;

parsing\_table[1][5] = 2;

parsing\_table[2][0] = 3;

parsing\_table[2][1] = 3;

parsing\_table[3][2] = 5;

parsing\_table[3][3] = 4;

parsing\_table[3][4] = 5;

parsing\_table[3][5] = 5;

parsing\_table[4][0] = 6;

parsing\_table[4][1] = 7;

로 채운다.

Init\_table은 1번과 같이 for문을 사용하여 -1로 초기화를 해준다.

Main은 1번과 같이

Init\_table을 load\_create\_rule 위에 호출한다.