**프로그래밍언어(가)**

**과제3**

**이름 : 허예림**

**학번 : 20182667**

**교수님 : 유재우 교수님**

**1. Calculator(수식게산기) program using recursive-descent parsing techniques of slides:24-25**

**(1) 소스 코드**

//

// main.c

// recursive\_descent parsing

//

// Created by yerimhuh on 2020/04/01.

// Copyright © 2020 yerimhuh. All rights reserved.

//

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_LENGTH 64

#define is\_num(c) ((c) >= '0' && (c) <= '9')

void get\_next\_token(void);

int expr(void);

int term(void);

int factor(void);

enum token{PLUS = 0,STAR,LP,RP,NUMBER,END};

enum token tok;

int input;

int r = 0;

int main(){

get\_next\_token();

r = expr();

printf("계산 결과 = %d\n",r);

while(tok == END){

printf("프로그램 끝\n");

exit(1);

}

}

void get\_next\_token(){

while((input = getchar()) != EOF){

if( input == ' ' || input == '\t')

continue;

break;

} // 공백과 탭은 무시하고 계산한다.

switch(input){

case '+':

tok = PLUS;

return ;

case '\*':

tok = STAR;

return ;

case '(':

tok = LP;

return ;

case ')':

tok = RP;

return ;

}

if(is\_num(input)){

input = input-'0';

tok = NUMBER;

return ;

}

if(input == 10){

fflush(stdin);

tok = END;

} // enter값 무시

}

int expr(){

r = term();

while(tok == PLUS){

r = term();

get\_next\_token();

r = r + term();

}

return r;

}

int term(){

r = factor();

while(tok == STAR){

r = factor();

get\_next\_token();

r = r \* factor();

}

return r;

}

void error(){

printf("error입니다. \n");

exit(1);

}

int factor(){

if(tok == NUMBER){

r = input;

get\_next\_token();

}

else if(tok == LP){

get\_next\_token();

r = expr();

if(tok == RP){

get\_next\_token();

r = expr();

}

else{

if(tok == END)

return r;

else

error();

}

}

else{

if(tok == PLUS || tok == STAR)

return r;

else

error();

}

return r;

}

**(2) 결과 캡쳐**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**2. Problem (연습문제) set of chapter-3 (pp.181-184) 2,3,4,6,7,8,11,14,17,19,20**

**(2) Write EBNF descriptions for the following:**

**a. A Java class definition header statement**

<class\_head> → {<modifier>} class <id> [extends class\_name]

[implements <interface\_name> {, <interface\_name>}]

<modifier> → public | abstract | final

**b. A Java method call statement**

<for> -> for ‘(‘ [[<type>] <id> = <expr> {, [<type>] <id> = <expr>}] ; [<expr>] ; [<expr>{, <expr>}] ‘)’ ‘{‘ <stmt\_list> ‘}’

**c. A C switch statement**

<switch\_stmt> → switch ( <expr> ) {case <literal> : <stmt\_list>

{case <literal> : <stmt\_list> } [default : <stmt\_list>] }

**d. A C union definition**

<union\_defn> -> union <var\_list> <union\_identifier>;

<var\_list> -> <list\_of\_data-type specifier> <var>

<list\_of\_data-type specifier> -> int | float | long |char | double

<union\_identifier> -> <var>

**e. C float literals**

 <float-literal> –>   <real> <suffix>

| <real> <exponent> <suffix>

| <integer> <exponent> <suffix>

**(3) Rewrite the BNF of Example 3.4 to give + precedence over \* and force + to be right associative.**

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <expr> \* <term> | <term>

<term> → <factor> + <term> | <factor>

<factor> → (<expr>) | <id>

**(4)** **Rewrite the BNF of Example 3.4 to add the ++ and -- unary operators of Java.**

<assign> -> <id> = <expr>   
<id> -> A | B | C   
<expr> -> <expr> + <term> | <term>   
<term> -> <term> \* <factor> | <factor>   
<factor> -> ( <expr> ) | <id> | <id> ++ | <id> - -

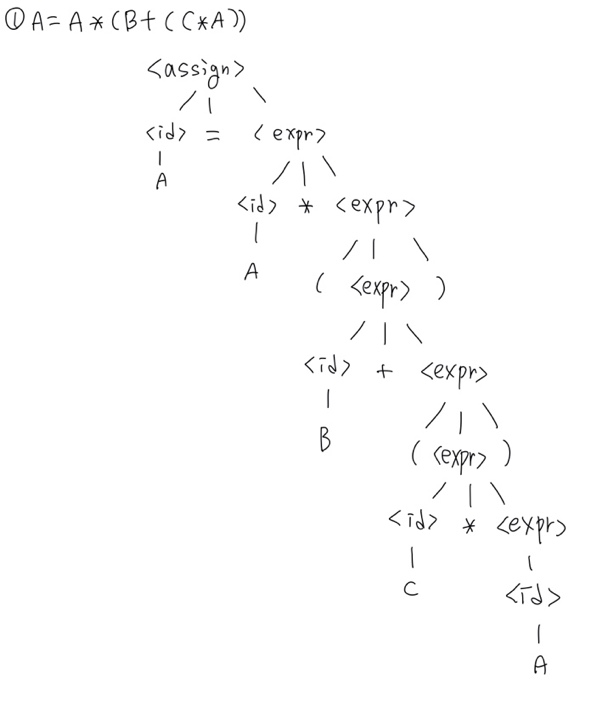
**(6) Using the grammar in Example 3.2, show a parse tree and a leftmost derivation for each of the following statements:**

**a. A = A \* (B + (C \* A))**

1) leftmost derivation

<assign> -> <id> = <expr>  
-> A = <expr>  
-> A = <id> \* <expr>   
-> A = A \* <expr>   
-> A = A \* (<expr>)  
-> A = A \* (<id> + <expr>)  
-> A = A \* (B + <expr>)    
-> A = A \* (B + (<expr>))  
-> A = A \* (B + (<id> \* <id>))  
-> A = A \* (B + ( C \* A))

2) parse tree

****

**b. B = C \* (A \* C + B)**

1) leftmost derivation

<assign> -> <id> = <expr>

-> B = <expr>

-> B = <id> \* <expr>

-> B = C \* <expr>

-> B = C \* (<expr>)

-> B = C \* (<id> \* <expr>)

-> B = C \* ( A \* <expr>)

-> B = C \* ( A \* <id> + <expr>)

-> B = C \* ( A \* C + <expr>)

-> B = C \* ( A \* C +<id>)

-> B = C \* ( A \* C + B )

2) parse tree

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**c. A = A \* (B + (C))**

1) leftmost derivation

<assign> -> <id> = <expr>

-> A = <expr>

-> A = <id> \* <expr>

-> A = A \* (expr>)

-> A = A \* (<id> + (<expr>))

-> A = A \* (B + <expr>)

-> A = A \* (B + (<expr>))

-> A = A \* (B + (<id>))

-> A = A \* (B + (C))

2) parse tree

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**(7) Using the grammar in Example 3.4, show a parse tree and a leftmost derivation for each of the following statements:**

**a. A = (A + B) \* C**

1) leftmost derivation

<assign> -> <id> = <expr>

-> A = <expr>

-> A = <term>

-> A = <term> \* <factor>

-> A = <factor> \* <factor>

-> A = <expr> \* <factor>

-> A = (<expr>) \* <factor>

-> A = (<expr> + <term>) \* <factor>

-> A = (<term> + <term>) \* <factor>

-> A = (<factor> + <term>) \* <factor>

-> A = (<id> + <term>) \* <factor>

-> A = (A + <term>) \* <factor>

-> A = (A + <factor>) \* <factor>

-> A = (A + <id>) \* <factor>

-> A = (A + B) \* <factor>

-> A = (A + B) \* <id>

-> A = (A + B) \* C

2) parse tree

**텍스트이(가) 표시된 사진

자동 생성된 설명  
b. A = B + C + A**

1) leftmost derivation

<assign> -> <id> = <expr>

-> A = <expr>

-> A = <expr> + <term>

-> A = <expr> + <term> + <term>

-> A = <term> + <term> + <term>

-> A = <factor> + <term> + <term>

-> A = <id> + <term> + <term>

-> A = B + <term> + <term>

-> A = B + <factor> + <term>

-> A = B + <id> + <term>

-> A = B + C + <term>

-> A = B + C + <factor>

-> A = B + C + <id>

-> A = B + C + A

2) parse tree

**텍스트이(가) 표시된 사진

자동 생성된 설명  
c. A = A \* (B + C)**

1) leftmost derivation

<assign> -> <id> = <expr>

-> A = <expr>

-> A = <term>

-> A = <term> \* <factor>

-> A = <factor> \* <factor>

-> A = <id> \* <factor>

-> A = A \* <factor>

-> A = A \* (<expr>)

-> A = A \* (<expr> + <term>)

-> A = A \* (<term> + <term>)

-> A = A \* (<factor> + <term>)

-> A = A \* (<id> + <term>)

-> A = A \* (B + <term>)

-> A = A \* (B + <factor>)

-> A = A \* (B + <id>)

-> A = A \* (B + C)

2) parse tree

**텍스트이(가) 표시된 사진

자동 생성된 설명  
d. A = B \* (C \* (A + B))**

1) leftmost derivation

<assign> -> <id> = <expr>

-> A = <expr>

-> A = <term>

-> A = <term> \* <factor>

-> A = <factor> \* <factor>

-> A = <id> \* <factor>

-> A = B \* <factor>

-> A = B \* (<expr>)

-> A = B \* (<term>)

-> A = B \* (<term> \* <factor>)

-> A = B \* (<factor> \* <factor>)

-> A = B \* (<id> \* <factor>)

-> A = B \* (C \* <factor>)

-> A = B \* (C \* (<expr>)

-> A = B \* (C \* (<expr> + <term>))

-> A = B \* (C \* (<term> + <term>))

-> A = B \* (C \* (<factor> + <term>))

-> A = B \* (C \* (<id> + <term>))

-> A = B \* (C \* (A + <term>))

-> A = B \* (C \* (A + <factor>))

-> A = B \* (C \* (A + <id>))

-> A = B \* (C \* (A + B))

2) parse tree

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**(8) Prove that the following grammar is ambiguous:**

**<S> → <A>**

**<A> → <A> + <A> | <id>**

**<id> → a | b | c**

1) <S> -> <A>

-> <A> + <A>

-> <id> + <A>

-> a + <A>

-> a + <A> + <A>

-> a + <id> + <A>

-> a + b + <A>

-> a + b + <id>

-> a + b + c

2) <S> -> <A>

-> <A> + <A>

-> <A> + <A> + <A>

-> <id> + <A> + <A>

-> a + <A> + <A>

-> a + <id> + <A>

-> a + b + <id>

-> a + b + c

=> 두 개의 다른 parse tree가 있으므로 ambiguous하다고 말할 수 있다.

**(11) Consider the following grammar:**

**<S> → <A> a <B> b**

**<A> → <A> b | b**

**<B> → b**

**Which of the following sentences are in the language generated by this grammar?**

**a. babb**

**b. bbbabb**

**c. bbaaaaabc**

**d. aaaaaa**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

답 : a,b

**(14) Draw parse trees for the sentences abb and aabbb, as derived from the grammar of Problem 13.**

Grammer of problem 13

<S> -> <A> b

<A> -> a<A>b | ab

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

**(17) Convert the following EBNF to BNF:**

**S → A{bA}**

**A → a[b]A**

S -> A | AB

B -> bA | bAB

A -> aA | abA

**(19) Write an attribute grammar whose BNF basis is that of Example 3.6 in Section 3.4.5 but whose language rules are as follows: Data types cannot be mixed in expressions, but assignment statements need not have the same types on both sides of the assignment operator.**

1) Syntax rule: <assign> → <var> = <expr>

2) Syntax rule: <expr> → <var>[2] + <var>[3]

Predicate: <var>[2].actual\_type == <var>[3].actual\_type

3) Syntax rule: <expr> → <var>

4) Syntax rule: <var> → A | B | C

Semantic rule: <var>.actual\_type ← lookup(<var>.string)

**(20) Write an attribute grammar whose base BNF is that of Example 3.2 and whose type rules are the same as for the assignment statement example of Section 3.4.5.**

1) syntax rule: <assign> → <id> = <expr>

semantic rule: <expr>.expected\_type ← <id>.actual\_type

predicate: <expr>[1].actual\_type == <expr>[1].expected\_type

2) syntax rule: <expr>[1] → <id> + <expr>[2]

semantic rule: <expr>[1].actual\_type ←

if (<id>.actual\_type = int and <expr>[2].actual\_type = int)

then int

else float

end if

3) syntax rule: <expr>[1] → <id> \* <expr>[2]

semantic rule: <expr>[1].actual\_type ←

if (<id>.actual\_type = int and <expr>[2].actual\_type = int)

then int

else float

end if

4) syntax rule: <expr>[1] → ( <expr>[2] )

semantic rule: <expr>[1].actual\_type ← <expr>[2].actual\_type

5) syntax rule: <expr> → <id>

semantic rule: <expr>.actual\_type ← <id>.actual\_type

6) syntax rule: <id> → A | B | C

semantic rule: <id>.actual\_type ← lookup(<

id>.string)

**3. Correctness proof(증명): Slide:34**

**텍스트이(가) 표시된 사진

자동 생성된 설명**