컴파일러 과제 2

yacc & lex program

제출마감일자 : 2016.10.10

담당교수 : 유재우 교수님

소속학부 : 컴퓨터학부

학번 : 20142577

이름 : 홍상원

1.1. yacc program 구현 : yoo.y

```
%{
#include "symbol.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*미리 정의된 함수들 extern 하여 사용*/
//token 읽고 actions 실행시키고 파일 끝을 만나거나 syntax error 발생 시 return
extern int yyparse (void);
//lex 를 생성하여 분석을 시작
extern int yylex();
//syntax error 발생 시 호출
void yyerror(const char *s);
%}
%union {
        int nval;
        struct symtab *pid;
}
%token EQL NL PLUS STAR LPAREN RPAREN
%token <nval> NUMBER
%token <pid> IDENT
%type <nval> factor term expression
%%
program : statement_list
                ;
statement_list : statement_list statement NL
                | statement NL
                ;
statement : IDENT EQL expression        { $1->value = $3; }
                | expression            { printf("%d₩n", $1); }
                ;
expression : expression PLUS term       { $$ = $1 + $3; }
                | term                  { $$ = $1; }
                ;
term : term STAR factor                 { $$ = $1 * $3; }
                | factor                { $$ = $1; }
                ;
factor : LPAREN expression RPAREN       { $$ = $2; }
                | IDENT                 { $$ = $1->value; }
```

```
                | NUMBER                { $$ = $1; }
                ;
%%
```

1.2. yoo.y 분석

- %union : semantic value 를 위한 데이터 공용체

- 기본적인 토큰 선언
%token EQL NL PLUS STAR LPAREN RPAREN

- NUMEBR 를 %union 의 nval type(int)으로 선언
%token <nval> NUMBER

- IDENT 를%union 의 pid type(struct symtab *)으로 선언
%token <pid> IDENT

- factor, term, expression 를 %union 의 nval 로 value type 정의
%type <nval> factor term expression

- BNF 표기법 : 연산 규칙
```
program : statement_list
                ;
statement_list : statement_list statement NL
                | statement NL
                ;
statement : IDENT EQL expression        { $1->value = $3; } : 대입 연산
                | expression            { printf("%d₩n", $1); }
                ;
expression : expression PLUS term       { $$ = $1 + $3; } : 덧셈 연산
                | term                  { $$ = $1; }
                ;
term : term STAR factor                 { $$ = $1 * $3; } : 곱셈 연산
                | factor                { $$ = $1; }
                ;
factor : LPAREN expression RPAREN       { $$ = $2; }
                | IDENT                 { $$ = $1->value; }
                | NUMBER                { $$ = $1; }
                ;
```

2.1. lex program 구현 : yoo.l

```
%{
#include <stdlib.h>
#include "symbol.h"
#include "y.tab.h"
int yywrap();//파일의 끝에서 호출
%}
digit    [0-9]
letter   [a-zA-Z_]
delim    [ ₩t]
line     [₩n]
ws       {delim}+
%%
{ws}                                    { }
{line}                                  { return (NL); }
"₩+"                                    { return (PLUS); }
"₩*"                                    { return (STAR); }
"₩("                                    { return (LPAREN); }
"₩)"                                    { return (RPAREN); }
"₩="                                    { return (EQL); }
{digit}+                                { yylval.nval=atoi(yytext); return (NUMBER); }
{letter}({letter}|{digit})*             { yylval.pid = symlook(yytext); return (IDENT); }
"//"[^₩n]*                              { }
%%
int yywrap() {
        return 1; //1 을 반환하면 파싱 정지
}
```

2.2. yoo.l 분석

- lex 명세서
digit [0-9] : digit 을 0 부터 9 사이의 정수로 매핑
letter [a-zA-Z_] : letter 를 a 부터 z 사이, A 부터 Z 사이, _ 문자로 매핑
delim [₩t] : delim 을 tab, 공백 문자로 매핑
line [₩n] : line 을 개행문자로 매핑
ws {delim}+ : ws 를 공백이 하나 이상 있는 경우로 매핑

{ws} { }
: 공백인 경우 아무 것도 안함
{line} { return (NL); }
: 개행문자인 경우 NL token 반환
"₩+" { return (PLUS); }
: + 인 경우 PLUS token 반환
"₩*" { return (STAR); }
: * 인 경우 STAR token 반환
"₩(" { return (LPAREN); }
: (인 경우 LPAREN token 반환
"₩)" { return (RPAREN); }
:) 인 경우 RPAREN token 반환
"₩=" { return (EQL); }
: = 인 경우 EQL token 반환
{digit}+ { yylval.nval=atoi(yytext); return (NUMBER); }
: 0 -9 사이 정수가 하나 이상있는 경우 yylval 변수에 yytext 를 정수로 변환하여 저장, 그리고 NUMBER token 반환, yylval 와 yytext 는 미리 정의된 변수
{letter}({letter}|{digit})* { yylval.pid = symlook(yytext); return (IDENT); }
: 문자이거나 문자, 정수가 조합된 형태인 경우 yylval 변수에 yytext 를 symbol table 형태로 저장, 그리고 IDENT token 반환, yylval 와 yytext 는 미리 정의된 변수
"//"[^₩n]* { }
: 주석이면 아무 것도 안함

3. symbol.h

```
#define NSYMS 100

//symbol table
struct symtab {
        char *name;
        double value;
} symtab[NSYMS];
struct symtab *symlook();//symbol table 찾기 함수
```

4. symbol.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "symbol.h"


//syntax error 함수 extern
void yyerror(char *s) {
	printf("%s₩n", s);
}

//symbol table 찾기
struct symtab *symlook(char *s)
{
	struct symtab *sp;
	for(sp=symtab; sp < &symtab[NSYMS]; sp++) {

		if(sp->name && !strcmp(sp->name, s)) //symbol table 에 존재하면
			return sp;

		if(!sp->name) {//symbol table 에 존재하지 않으면
			sp->name=strdup(s); //새로운 symbol 생성
			/*strdup() : 복사할 문자열 크기에 맞는 메모리를 확보한 후
			문자열을 복사한 후, 확보한 메모리의 포인터를 반환*/
			return sp;
		}

	}
	//symbol table 용량 초과
	yyerror("Too many symbols");
	exit(1);
}
```

5. main.c

```
//token 읽고 actions 실행시키고 파일 끝을 만나거나 syntax error 발생 시 return
extern int yyparse (void);
int main() {

	return yyparse();
}
```

6. 실행결과

- yacc -d yoo.y 결과 y.tab.h 와 y.tab.c 생성

```
[hongsang-won-ui-MacBook-Pro:Sample2 Frodo$ yacc -d yoo.y
[hongsang-won-ui-MacBook-Pro:Sample2 Frodo$ ls
main.c          symbol.h        y.tab.h         yoo.y
symbol.c        y.tab.c         yoo.l
```

- lex yoo.l 결과 lex.yy.c

```
[hongsang-won-ui-MacBook-Pro:Sample2 Frodo$ lex yoo.l
[hongsang-won-ui-MacBook-Pro:Sample2 Frodo$ ls
lex.yy.c        symbol.c        y.tab.c         yoo.l
main.c          symbol.h        y.tab.h         yoo.y
```

- 컴파일 후 실행 결과

```
[hongsang-won-ui-MacBook-Pro:Assignment2 Frodo$ gcc lex.yy.c y.tab.c y.tab.h main]
.c symbol.c symbol.h
[hongsang-won-ui-MacBook-Pro:Assignment2 Frodo$ ./a.out                          ]
3 + 7
10
6 * 7
42
9 * (10 + 2)
108
```