```
%{
#include <stdio.h>
#include <string.h>
#include "y.tab.h"

extern FILE *yyin;
char *FILE_name;
```

FILE_name : 파일 이름 저장하는 변수

```
void yyerror(const char *str)
{
  extern int yylineno;
  extern char *yytext;
  fprintf(stderr, "%s:%d: error: %s '%s' token\n", FILE_name, yylineno, str,
yytext);
}
```

error 발생 시 FILE_name 출력, yylineno에는 에러 발생 시 몇 번째 줄인지 저장되어 있다. Str에 무슨 에러인지 적혀있다. yytext에는 에러 발생 시의 토큰이 저장되어 있다.

```
int yywrap() {
  return 1;
}
```

파일 끝나면 종료

```
int main(argc, argv)
int argc;
char **argv;
{
       ++argv, --argc;
       if( argv > 0 ) {
              yyin = fopen(argv[0], "r");
              FILE_name = argv[0];
```

FILE_name에 파일 이름 저장

```
              }
       else
              yyin = stdin;
       yyparse();
       return 0;
}

%}

%token DEFINE INT VOID IF FOR CONTINUE OP_ASSIGN OP_INC OP_ADD OP_MUL
OP_LOGIC OP_REL
```

토큰 선언

```
%union
{
  int number;
  char *string;
}
%token <number> NUM
%token <string> ID
```

NUM ID 토큰에 type 부여

```
%type <string> variable
%type <string> assignments
```

Non-term의 type 부여

```
%%
program: /* empty */
    | program line
    ;

line:
    header
    | function
    | fline
    ;

fline:
    expression
    | statement
    | clause
    ;
```

function 안에 들어갈 수 있는 line들은 fline으로 분리했다. Header와 function의 경우 function안에 들어갈 수 없다.

```
header:
    DEFINE ID NUM
    { printf("Header(Define)\n"); }
    ;
```

header는 DEFINE ID NUM에 대해서만 고려.

```
function:
    type ID '(' func_declarations ')' '{' contents '}'
    { printf("Function (%s) matched!\n", $2); }
    ;
```

위와 같은 조건을 만족하면 function matched 출력하는데, $2를 이용해 두 번째 토큰인 ID의 값을 출력한다.

```
type:
    VOID
    | INT
    ;
```

```
func_declarations: /* empty */
    | declarations
    { printf("Expression(FunctionArgsDec)\n"); }
    ;
```

function 안에서 선언이 된다면 FunctionArgsDec 출력을 한다.

```
declarations:
    declarations ',' declaration
    | declaration
    ;
```

하나의 dec또는 콤마(,)로 분리된 여러 개의 dec가 있을 수 있다. Ex) int x, int y

```
declaration:
    type variable
    ;
```

type과 변수로 이루어진다.

```
variable:
    ID arrays
    { $$=$1; }
    ;

arrays:
    | arrays array
    ;
array:
    '[' ']'
    | '[' expression ']'
    ;
```

배열일 경우 뒤에 ID 뒤에 '[' , ']' 가 여러 개 붙을 수 있으며 그 안에 expression이 들어갈 수 있다. $$=$1;을 이용해 variable안에 ID의 값을 저장했다.

```
expression:
    factor
    | factor OP_AM expression
    ;
OP_AM:
    OP_ADD
    | OP_MUL
    ;
factor:
    variable
    | NUM
    ;
```

여러 expression을 체크한다. Ex) x+2*y

```
contents:
    | contents fline
    ;
```

함수 안에 들어가는 contents는 header와 function이 들어갈 수 없다.

```
multi_declaration:
    type variables
    ;
variables:
    variable
    | variables ',' variable
    ;
```

변수를 동시에 여러 개 선언할 수 있다. Ex) int x, y, c;

```
statement:
    error ';'
    | multi_declaration ';'
    { printf("Statement(Declaration)\n"); }
    | assignments ';'
    { printf("Statement(Assignment,%s)\n", $1); }
    | CONTINUE ';'
    { printf("Statement(Continue)\n"); }
    ;
```

각각의 non-term마다 그에 맞는 출력을 하도록 한다. Assignments의 경우 변수 이름을 출력한다. error가 발생하면 semicolon전까지 token을 무시하고 진행한다.

```
assignments:
    variable OP_ASSIGN expression
    { $$=$1; }
    ;
```

variable 값을 assignments에 저장한다.

```
clause:
    FOR '(' InitializeStatement ';' TestExpressions ';' UpdateStatement ')'
content
    { printf("Clause(for) matched!\n"); }
    | IF '(' TestExpressions ')' content
    { printf("Clause(if) matched!\n"); }
    ;
```

For문과 IF문.

```
content:
    '{' contents '}'
    | fline
    ;
```

For문과 IF문 다음에 '{' '}'이 온다면 여러 줄이 올 수 있지만 없다면 한 줄 밖에 오지 못 한다.

```
InitializeStatement:
    | assignments
    { printf("Statement(Initialize)\n"); }
    ;
```

Assignments와 같다. 그러나 For문의 시작이므로 Initialize문구를 출력한다.

```
TestExpressions:
    | TestExpression
    { printf("Expression(Test)\n"); }
    ;
TestExpression:
    TestFactor
    | TestFactor OP_LOGIC TestExpression
    ;
TestFactor:
    expression OP_REL expression
    | expression
    ;
```

Test에 사용되는 조건 expression non-term과 같은 구조다.

```
UpdateStatement:
    | ID OP_INC
    { printf("Statement(Update)\n"); }
    ;
%%
```

For문의 맨 뒤에 오는 증가 조건의 경우 Update 출력한다.