REPORT

“ Parser 구현 ”

과 목 명	컴파일러 설계 및 구축 (월25,26)
담당교수	이양선 교수님
학 과	컴퓨터공학과
학 번	2015305084
이 름	홍송희
제 출 일	2018.11.19

<코드>

main.c

parser | tokenType

```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include "minic.h"

#define PS_SIZE 200

#define NO_KEYWORDS 7
#define ID_LENGTH 12

void parser(FILE* source_file);
void semantic(int);
void printToken(struct tokenType token); //EERROR
void dumpStack();   //EERROR
void errorRecovery(FILE* source_file); //EERROR

int sp;
int stateStack[PS_SIZE];
int symbolStack[PS_SIZE];

void lexicalError(int n);
int superLetter(char ch);
int superLetterOrDigit(char ch);
int getIntNum(char firstCharacter, FILE* source_file);

char id[ID_LENGTH];
char ch;

char *keyword[NO_KEYWORDS] = { "const", "else", "if","int","return","void","while" };

struct tokenType {
    int number;
    union {
        char id[ID_LENGTH];
        int num;
    }value;
};

enum tsymbol {
    tnull = -1,
    tnot, tnotequ, tmod, tmodAssign, tident, tnumber,
    tand, tlparen, trparen, tmul, tmulAssign, tplus,
    tinc, taddAssign, tcomma, tminus, tdec, tsubAssign,
    tdiv, tdivAssign, tsemicolon, tless, tlesse, tassign,
    tequal, tgreat, tgreate, tlbracket, trbracket, teof,
    tconst, telse, tif, tint, treturn, tvoid,
    twhile, tlbrace, tor, trbrace
};

enum tsymbol tnum[NO_KEYWORDS] = { tconst, telse, tif, tint, treturn, tvoid, twhile };

struct tokenType scanner(FILE* source_file)
{
    struct tokenType token;
    int i, index;


    token.number = tnull;

    do {
        while (isspace(ch = fgetc(source_file)));
        if (superLetter(ch)) {
            i = 0;
            do {
                if (i < ID_LENGTH)  id[i++] = ch;
                ch = fgetc(source_file);
            } while (superLetterOrDigit(ch));
            if (i >= ID_LENGTH) lexicalError(1);
            id[i] = '₩0';
            ungetc(ch, stdin);
            for (index = 0; index < NO_KEYWORDS; index++)
                if (!strcmp(id, keyword[index]))    break;
            if (index < NO_KEYWORDS)
                token.number = tnum[index];
            else {
                token.number = tident;
                strcpy_s(token.value.id, ID_LENGTH, id);
            }
        }
```

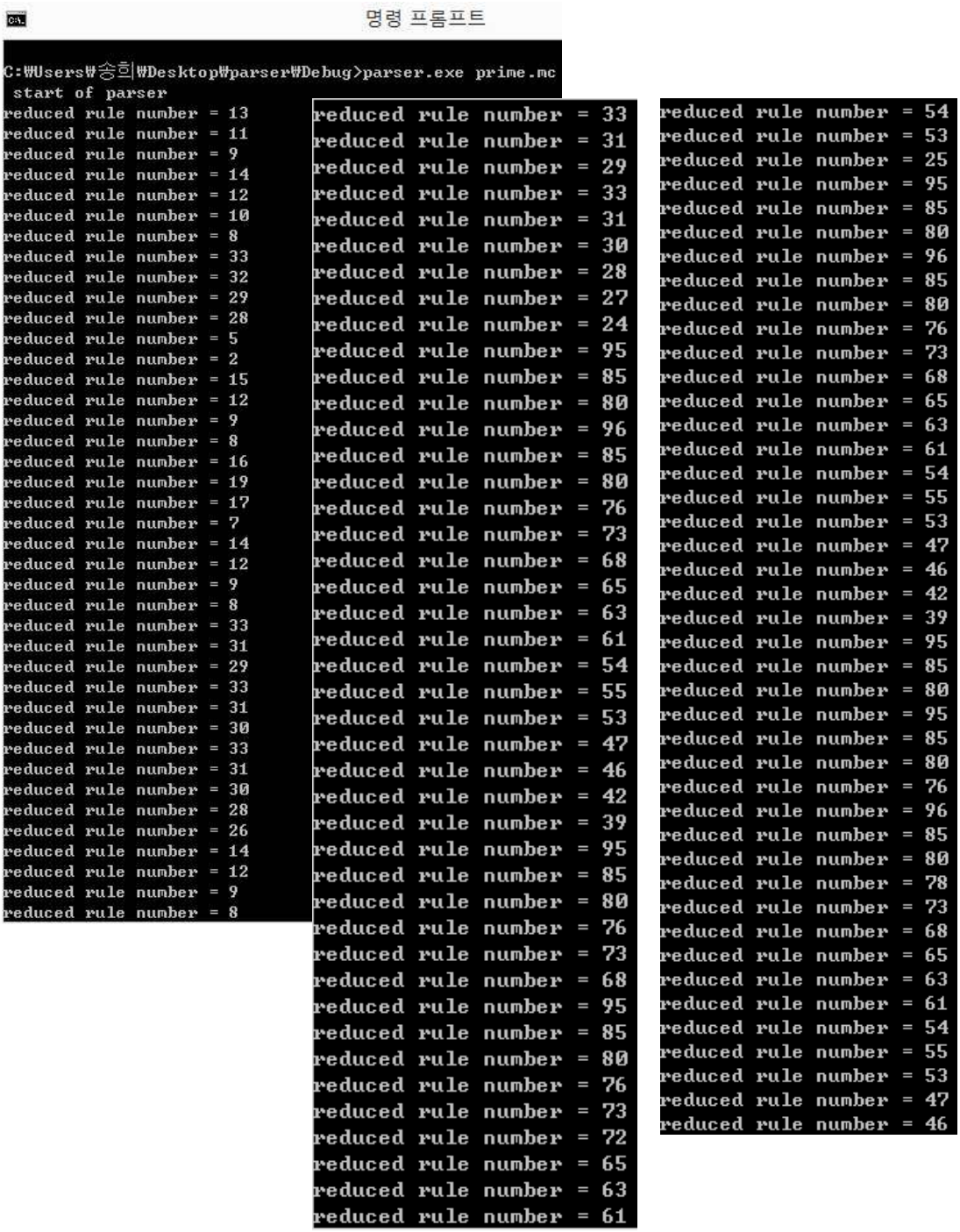
```
        else if (isdigit(ch)) {
            token.number = tnumber;
            token.value.num = getIntNum(ch, source_file);
        }
        else {
            switch (ch) {
            case '/':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '*')
                    do {
                        while (ch != '*')   ch = fgetc(source_file);
                        ch = fgetc(source_file);
                    } while (ch != '/');
                else if (ch == '/')
                    while (fgetc(source_file) != '\n');
                else if (ch == '=') token.number = tdivAssign;
                else {
                    token.number = tdiv;
                    ungetc(ch, stdin);
                }
                break;
            case '!':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '=') { token.number = tnotequ; id[1] = ch; }
                else {
                    token.number = tnot;
                    ungetc(ch, stdin);
                }
                break;
            case '%':
                id[0] = ch;
                ch = fgetc(source_file);
                if (ch == '=') {
                    token.number = tmodAssign;
                    id[1] = ch;
                }
                 break;
             case '(':   id[0] = ch; token.number = tlparen; break;
             case ')':   id[0] = ch; token.number = trparen; break;
             case ',':   id[0] = ch; token.number = tcomma;  break;
             case ';':   id[0] = ch; token.number = tsemicolon;  break;
             case '[':   id[0] = ch; token.number = tlbracket;   break;
             case ']':   id[0] = ch; token.number = trbracket;   break;
             case '{':   id[0] = ch; token.number = tlbrace; break;
             case '}':   id[0] = ch; token.number = trbrace; break;
             case EOF:   token.number = teof;    break;
             default: {
                 printf("Current character : %c", ch);
                 lexicalError(4);
                 break;
             }
             }

        }
    } while (token.number == tnull);

    return token;
}

void lexicalError(int n) {
    printf(" *** Lexical Error : ");
    switch (n) {
    case 1: printf("an identifier length must be less than 12.\n");
        break;
    case 2: printf("next character must be &.\n");
        break;
    case 3: printf("next character must be |.\n");
        break;
    case 4: printf("invalid character!!!\n");
        break;
    }
}

int superLetter(char ch) {
    if (isalpha(ch) || ch == '_')   return 1;
    else return 0;
}
```

```
        else {
            token.number = tmod;
            ungetc(ch, stdin);
        }
        break;
    case '&':
        id[0] = ch;
        ch = fgetc(source_file);
        if (ch == '&') { token.number = tand; id[1] = ch; }
        else {
            lexicalError(2);
            ungetc(ch, stdin);
        }
        break;
    case '*':
        id[0] = ch;
        ch = fgetc(source_file);
        if (ch == '=') { token.number = tmulAssign; id[1] = ch; }
        else {
            token.number = tmul;
            ungetc(ch, stdin);
        }
        break;
    case '+':
        id[0] = ch;
        ch = fgetc(source_file);
        if (ch == '+') { token.number = tinc;  id[1] = ch; }
        else if (ch == '=') { token.number = taddAssign; id[1] = ch; }
        else {
            token.number = tplus;
            ungetc(ch, stdin);
        }
        break;
    case '-':
        id[0] = ch;
        ch = fgetc(source_file);
        if (ch == '-') { token.number = tdec; id[1] = ch; }
        else if (ch == '=') { token.number = tsubAssign; id[1] = ch; }
        else {
            token.number = tminus;
            ungetc(ch, stdin);

    }
    break;
case '<':
    id[0] = ch;
    ch = fgetc(source_file);
    if (ch == '=') { token.number = tlesse; id[1] = ch; }
    else {
        token.number = tless;
        ungetc(ch, stdin);
    }
    break;
case '=':
    id[0] = ch;
    ch = fgetc(source_file);
    if (ch == '=') { token.number = tequal; id[1] = ch; }
    else {
        token.number = tassign;
        ungetc(ch, stdin);
    }
    break;
case '>':
    id[0] = ch;
    ch = fgetc(source_file);
    if (ch == '=') { token.number = tgreate; id[1] = ch; }
    else {
        token.number = tgreat;
        ungetc(ch, stdin);
    }
    break;
case '|':
    id[0] = ch;
    ch = fgetc(source_file);
    if (ch == '|') { token.number = tor; id[1] = ch; }
    else {
        lexicalError(3);
        ungetc(ch, stdin);
    }
    break;
```

```c
int superLetterOrDigit(char ch) {
    if (isalnum(ch) || ch == '_')   return 1;
    else return 0;
}

int getIntNum(char firstCharacter, FILE* source_file) {
    int num = 0;
    char ch;
ch = firstCharacter;
    do {
        num = 10 * num + (int)(ch - '0');
        ch = fgetc(source_file);
    } while (isdigit(ch));

    ungetc(ch, stdin);
    return num;
}

void parser(FILE* source_file) {
    extern int parsingTable[NO_STATES][NO_SYMBOLS + 1];
    extern int leftSymbol[NO_RULES + 1], rightLength[NO_RULES + 1];
    int errcnt = 0;
    int entry, ruleNumber, lhs;
    int currentState;
    struct tokenType token;

    sp = 0; stateStack[sp] = 0;
    token = scanner(source_file);
    while (1) {
        currentState = stateStack[sp];
        entry = parsingTable[currentState][token.number];
        if (entry > 0) {
            sp++;
            if (sp > PS_SIZE) {
                printf("critical compiler error : parsing stack overflow");
                exit(1);
            }
            symbolStack[sp] = token.number;
            stateStack[sp] = entry;
            token = scanner(source_file);
        }
        else if (entry < 0) {
            ruleNumber = -entry;
            if (ruleNumber == GOAL_RULE) {
                if (errcnt == 0) printf(" *** valid source *** \n");
                else printf(" *** error in source : %d\n", errcnt);
                return;
            }
            semantic(ruleNumber);
            sp -= rightLength[ruleNumber];
            lhs = leftSymbol[ruleNumber];
            currentState = parsingTable[stateStack[sp]][lhs];
            sp++;
            symbolStack[sp] = lhs;
            stateStack[sp] = currentState;
        }
        else {
            printf(" === error in source === \n");
            errcnt++;
            printf("Current Token : ");
            printToken(token);
            dumpStack();
            errorRecovery(source_file);
            token = scanner(source_file);
        }
    }
}

void semantic(int n) {
    printf("reduced rule number = %d\n", n);
}


void main(int argc, char *argv[]) {
    FILE* source_file;

    int i;
    struct tokenType token;

    if (argc != 2) {
    fprintf(stderr, "Usage : scanner <source file name>\n");
    exit(1);
```

```
    }

    if ((source_file = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "%s file not found \n", argv[1]);
    exit(-1);
    }


    printf(" start of parser \n");
    parser(source_file);
    printf(" end of parser \n");
    fclose(source_file);

}

char *tokenName[] = {
    "!",     "!=",    "%",     "%=",    "%ident",   "%number",
    "&&",    "(",     ")",     "*",     "*=",       "+",
    "++",    "+=",    ",",     "-",     "--",       "-=",
    "/",     "/=",    ";",     "<",     "<=",       "=",
    "==",    ">",     ">=",    "[",     "]",        "eof",
    "const","else",  "if",    "int",   "return",   "void",
    "while","{",     "||",    "}"
};

void printToken(struct tokenType token) {
    if (token.number == tident)
        printf("%s", token.value.id);
    else if (token.number == tnumber)
        printf("%d", token.value.num);
    else
        printf("%s", tokenName[token.number]);
}

void dumpStack() {
    int i, start;

    if (sp > 10) start = sp - 10;
    else start = 0;

    printf("\n *** dump state stack : ");
    for (i = start; i <= sp; ++i)
        printf(" %d", stateStack[i]);
    printf("\n *** dump symbol stack : ");
    for (i = start; i <= sp; ++i)
        printf(" %d", symbolStack[i]);
    printf("\n");
}

void errorRecovery(FILE* source_file ) {
    struct tokenType tok;
    int parenthesisCount, braceCount;
    int i;

    parenthesisCount = braceCount = 0;
    while (1) {
        tok = scanner(source_file);
        if (tok.number == teof) exit(1);
        if (tok.number == tlparen) parenthesisCount++;
        else if (tok.number == trparen) parenthesisCount--;
        if (tok.number == tlbrace) braceCount++;
        else if (tok.number == trbrace) braceCount--;
        if ((tok.number = tsemicolon) && (parenthesisCount <= 0) && (braceCount <= 0))
            break;
        for (i = sp; i >= 0; i++) {
            if (stateStack[i] == 36) break;
            if (stateStack[i] == 24) break;
            if (stateStack[i] == 25) break;
            if (stateStack[i] == 17) break;
            if (stateStack[i] == 2) break;
            if (stateStack[i] == 0) break;
        }
        sp = i;
    }

}
```

<실행 결과>


명령 프롬프트
C:₩Users₩송희₩Desktop₩parser₩Debug>parser.exe prime.mc
start of parser
reduced rule number = 13
reduced rule number = 11
reduced rule number = 9
reduced rule number = 14
reduced rule number = 12
reduced rule number = 10
reduced rule number = 8
reduced rule number = 33
reduced rule number = 32
reduced rule number = 29
reduced rule number = 28
reduced rule number = 5
reduced rule number = 2
reduced rule number = 15
reduced rule number = 12
reduced rule number = 9
reduced rule number = 8
reduced rule number = 16
reduced rule number = 19
reduced rule number = 17
reduced rule number = 7
reduced rule number = 14
reduced rule number = 12
reduced rule number = 9
reduced rule number = 8
reduced rule number = 33
reduced rule number = 31
reduced rule number = 29
reduced rule number = 33
reduced rule number = 31
reduced rule number = 30
reduced rule number = 33
reduced rule number = 31
reduced rule number = 30
reduced rule number = 28
reduced rule number = 26
reduced rule number = 14
reduced rule number = 12
reduced rule number = 9
reduced rule number = 8
reduced rule number = 33
reduced rule number = 31
reduced rule number = 29
reduced rule number = 33
reduced rule number = 31
reduced rule number = 30
reduced rule number = 28
reduced rule number = 27
reduced rule number = 24
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 96
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 55
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 39
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 72
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 25
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 96
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 55
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 39
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 96
reduced rule number = 85
reduced rule number = 80
reduced rule number = 78
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 55
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46

```
reduced rule number = 42
reduced rule number = 40
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 96
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 55
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 40
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 72
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 25
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 95
```

```
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 79
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 55
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 39
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 96
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 66
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 96
reduced rule number = 85
```

```
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 55
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 49
reduced rule number = 43
reduced rule number = 40
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 83
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 40
reduced rule number = 37
reduced rule number = 23
reduced rule number = 41
reduced rule number = 51
reduced rule number = 44
reduced rule number = 40
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
```

```
reduced rule number = 68
reduced rule number = 66
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 95
reduced rule number = 85
reduced rule number = 96
reduced rule number = 85
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 93
reduced rule number = 92
reduced rule number = 90
reduced rule number = 87
reduced rule number = 80
reduced rule number = 76
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 49
reduced rule number = 43
reduced rule number = 40
reduced rule number = 95
reduced rule number = 85
reduced rule number = 80
reduced rule number = 83
reduced rule number = 76
```

```
reduced rule number = 73
reduced rule number = 68
reduced rule number = 65
reduced rule number = 63
reduced rule number = 61
reduced rule number = 54
reduced rule number = 53
reduced rule number = 47
reduced rule number = 46
reduced rule number = 42
reduced rule number = 40
reduced rule number = 37
reduced rule number = 23
reduced rule number = 41
reduced rule number = 51
reduced rule number = 44
reduced rule number = 40
reduced rule number = 37
reduced rule number = 23
reduced rule number = 6
reduced rule number = 4
reduced rule number = 3
reduced rule number = 1
 *** valid source ***
 end of parser
```