어휘 분석기 결과 보고서

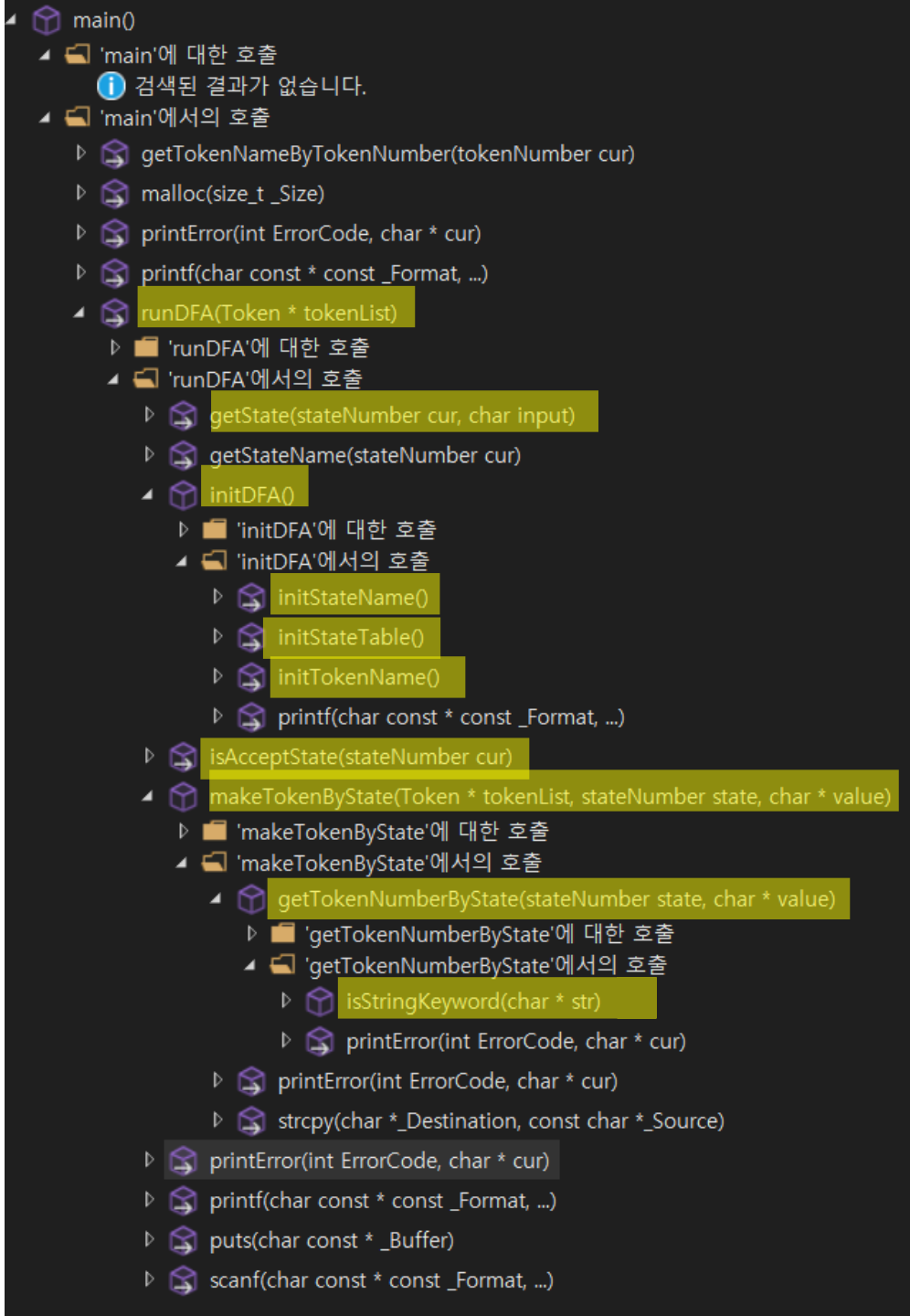
1891179 김현학

프로그램 호출 구조

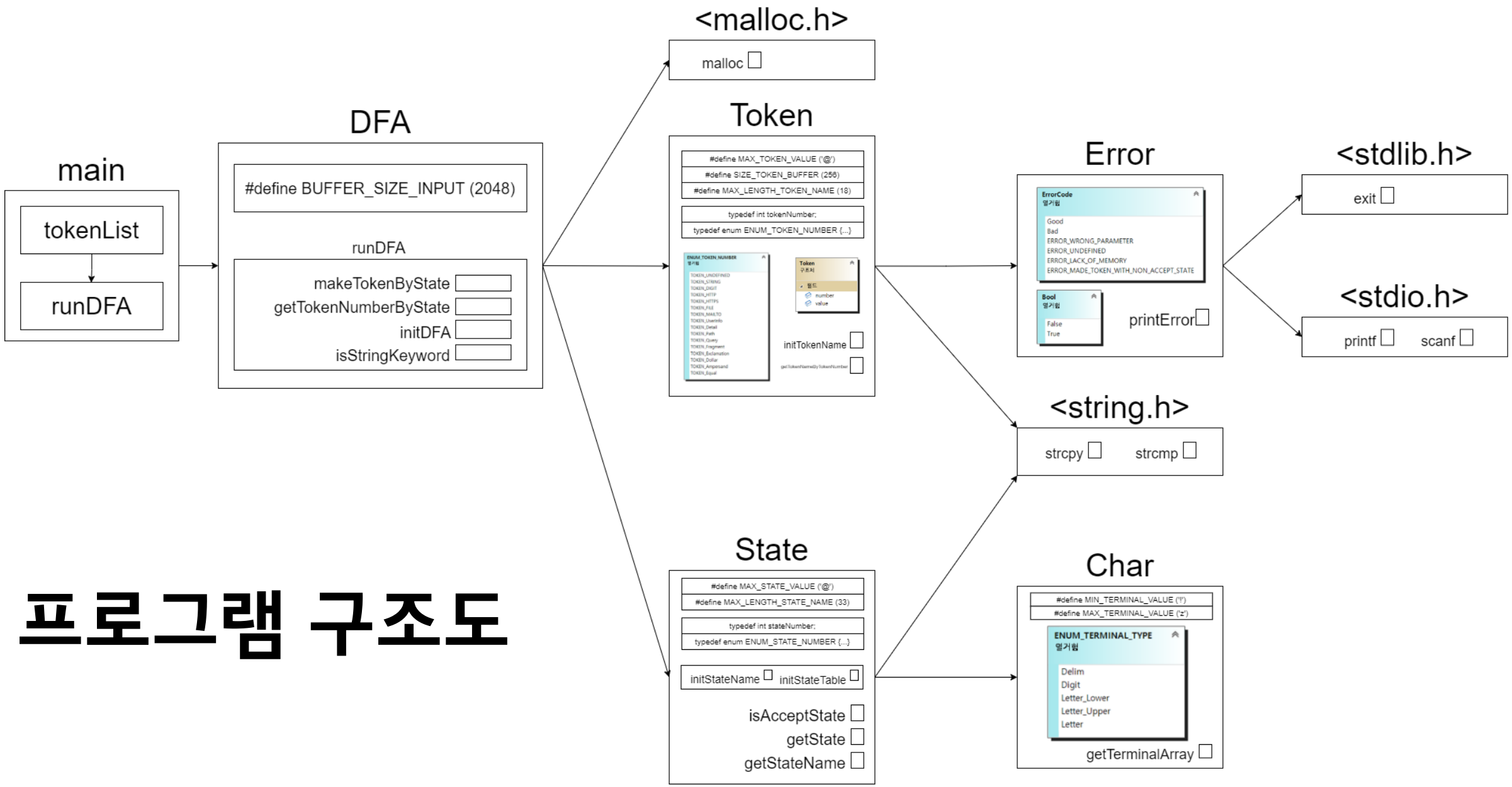
1. **Main** 함수에서
   Token 구조체의 배열 포인터를 넘겨주면

2. **runDFA** 함수를 통해입력 문자열에 대한
   TokenList 생성되고
   생성된 **Token**의 개수가 반환됨

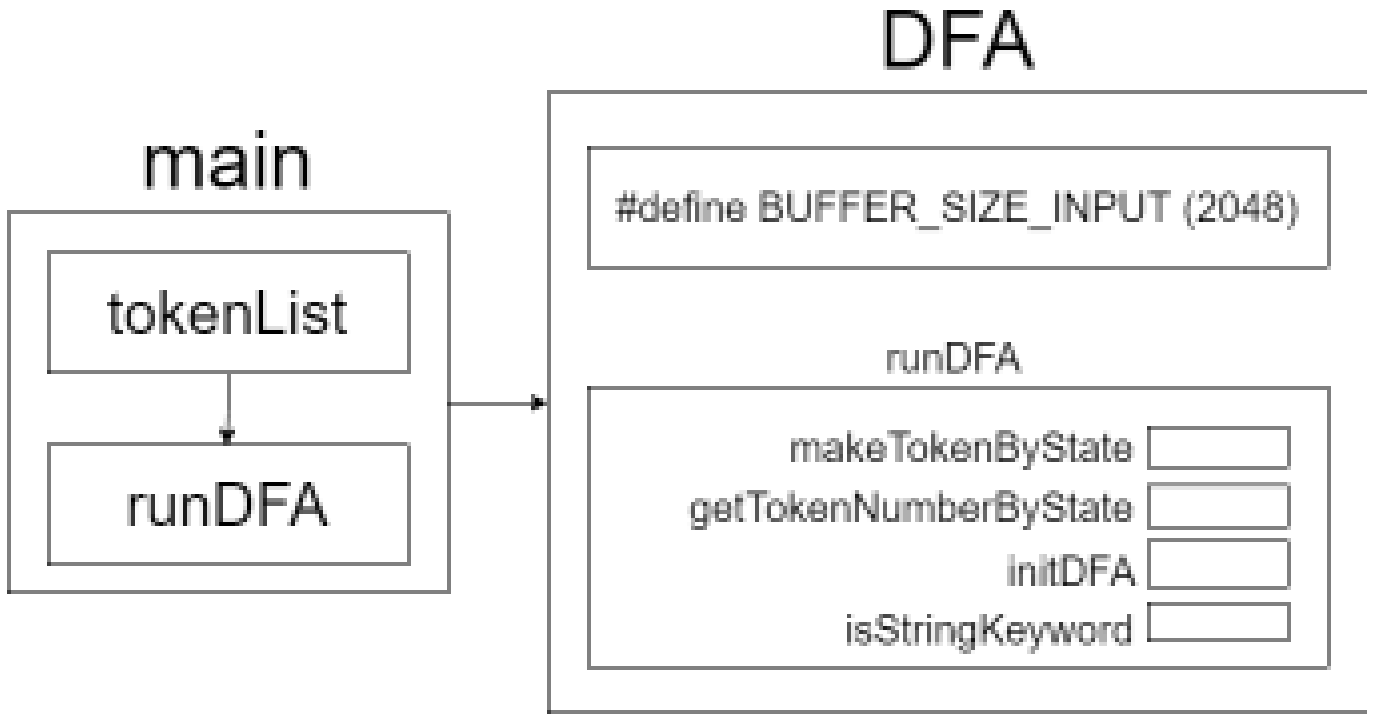
3. **For** 반복문을 활용하여
   Token의 개수만큼 **TokenList** 순회



프로그램 구조도


main
tokenList
runDFA
DFA
#define BUFFER_SIZE_INPUT (2048)
runDFA
makeTokenByState
getTokenNumberByState
initDFA
isStringKeyword
<malloc.h>
malloc
Token
#define MAX_TOKEN_VALUE ('@')
#define SIZE_TOKEN_BUFFER (256)
#define MAX_LENGTH_TOKEN_NAME (18)
typedef int tokenNumber;
typedef enum ENUM_TOKEN_NUMBER {...}
initTokenName
Error
ErrorCode
Good
Bad
ERROR_WRONG_PARAMETER
ERROR_UNDEFINED
ERROR_LACK_OF_MEMORY
ERROR_MADE_TOKEN_WITH_NON_ACCEPT_STATE
Bool
False
True
printError
<stdlib.h>
exit
<stdio.h>
printf
scanf
<string.h>
strcpy
strcmp
State
#define MAX_STATE_VALUE ('@')
#define MAX_LENGTH_STATE_NAME (33)
typedef int stateNumber;
typedef enum ENUM_STATE_NUMBER {...}
initStateName
initStateTable
isAcceptState
getState
getStateName
Char
#define MIN_TERMINAL_VALUE ('!')
#define MAX_TERMINAL_VALUE ('z')
ENUM_TERMINAL_TYPE
Delim
Digit
Letter_Lower
Letter_Upper
Letter
getTerminalArray

main – { DFA }


DFA
main
tokenList
runDFA
#define BUFFER_SIZE_INPUT (2048)
runDFA
makeTokenByState
getTokenNumberByState
initDFA
isStringKeyword

DFA – { Token, malloc, State}

Token

#define MAX_TOKEN_VALUE ('@')

#define SIZE_TOKEN_BUFFER (256)

#define MAX_LENGTH_TOKEN_NAME (18)

typedef int tokenNumber;

typedef enum ENUM_TOKEN_NUMBER {...}

initTokenName

getTokenNameByTokenNumber

<malloc.h>

malloc

State

#define MAX_STATE_VALUE ('@')

#define MAX_LENGTH_STATE_NAME (33)

typedef int stateNumber;

typedef enum ENUM_STATE_NUMBER {...}

initStateName initStateTable

isAcceptState

getState

getStateName

{ Token, State } – {Char, Error, string}

```c
// main.c

#define _CRT_SECURE_NO_WARNINGS
#include "DFA.h"

int main()
{
    struct Token* tokenList = (struct Token*)malloc(sizeof(struct Token) * 1000);
    if (tokenList == NULL) printError(ERROR_LACK_OF_MEMORY, "main()");

    int nToken = runDFA(tokenList);
    printf("nToken: %d\n", nToken);

    struct Token* tList = tokenList;
    for (int i = 0; i < nToken; ++i)
    {
        if (tList == NULL) break;
        char* tokenName = getTokenNameByTokenNumber(tList->number);
        printf("%s (%d, \"%s\")\n", tokenName, tList->number, tList->value);

        tList++;
    }

    return Good;
}
```

```c
// 결과로 만들어진 토큰의 개수를 반환한다.
int runDFA(struct Token* tokenList)
{
    char buf[BUFFER_SIZE_INPUT] = { 0, }; // input
    char tmp[BUFFER_SIZE_INPUT] = { 0, }; // lexeme

    char* str = buf;
    char* lexeme = tmp;

    while (!scanf("%[^\n]s", str)); // line feed parsing
    puts(str); // parsing result

    int nToken = 0;
    stateNumber next, cur = initDFA();

    for (char* c = str; ; ++c)
    {
        printf("    %c\t", *c);
        printf("%20s >>", getStateName(cur));
        cur = getState(cur, *c);
        printf("%20s\n", getStateName(cur));

        if (cur == STATE_EOF) break;

        *lexeme = *c; lexeme++;

        next = getState(cur, *(c + 1));
        if (isAcceptState(next)) {
            printf("\t%20s >>", getStateName(cur));
            cur = next;
            printf("\t%20s\n", getStateName(cur));
        }
        if ( isAcceptState(cur) )
        {
            *lexeme = '\0'; lexeme = tmp;
            tokenList = makeTokenByState(tokenList, cur, lexeme);
            nToken++;
            printf("\t%20s >>", getStateName(cur));
            cur = STATE_START;
            printf("\t%20s\n\n", getStateName(cur));
        }
    }
    if (nToken == 0) printError(ERROR_WRONG_PARAMETER, "runDFA(struct Token* tokenList)");
    else return nToken;
}
```

stateNumber initDFA()

```
{
    printf("initStateName....\t"); initStateName();     printf("finished!\n");
    printf("initTokenName....\t"); initTokenName();     printf("finished!\n");

    printf("initStateTable...\t"); initStateTable(); printf("finished!\n");
    // printf("initTokenTable...\t"); initTokenTable(); printf("finished!\n");
    // tokenTable 없이 hard-wired 방식으로 구현함

    return STATE_START;
}
```

void initStateTable()

```
{
    stateTable[STATE_START ]['\0'] = '\0';
    stateTable[STATE_STRING]['\0'] = STATE_ACC_STRING;
    stateTable[STATE_DIGIT ]['\0'] = STATE_ACC_DIGIT;

    // --------------
    //  GENERAL FORM
    // --------------
    char* str;

    // Letter
    str = getTerminalArray(Letter);
    printf("\n\nLetter: %s", str);
    for (int i = 0; str[i] != '\0'; ++i)
    {
        stateTable[STATE_START][str[i]] = STATE_STRING;
        stateTable[STATE_STRING][str[i]] = STATE_STRING;
        stateTable[STATE_DIGIT][str[i]] = STATE_STRING;

        puts("");
        log(STATE_START, str[i], STATE_DIGIT);
        log(STATE_DIGIT, str[i], STATE_DIGIT);
        log(STATE_STRING, str[i], STATE_STRING);
    }
    puts("");

    // DIGIT
    str = getTerminalArray(Digit);
    printf("Digit: %s", str);
    for (int i = 0; str[i] != '\0'; ++i) {
        stateTable[STATE_START][str[i]] = STATE_DIGIT;
        stateTable[STATE_STRING][str[i]] = STATE_STRING;
        stateTable[STATE_DIGIT][str[i]] = STATE_DIGIT;

        puts("");
        log(STATE_START, str[i], STATE_DIGIT);
        log(STATE_STRING, str[i], STATE_STRING);
        log(STATE_DIGIT, str[i], STATE_DIGIT);
    }
    puts("");



    // Delim
    str = getTerminalArray(Delim);
    printf("Delim: %s", str);
    for (int i = 0; str[i] != '\0'; ++i)
    {
        stateTable[STATE_STRING][str[i]] = STATE_ACC_STRING;
        stateTable[STATE_DIGIT][str[i]] = STATE_ACC_DIGIT;
        stateTable[STATE_START][str[i]] = str[i];

        puts("");
        log(STATE_STRING, str[i], STATE_ACC_STRING);
        log(STATE_DIGIT, str[i], STATE_ACC_DIGIT);
        log(STATE_START, str[i], str[i]);
    }
    puts("");
}
```

```
void initStateName()
{
    for (int i = 0; i <= MAX_STATE_VALUE; ++i)
        strcpy(stateName[i], "STATE_UNDEFINED");

    strcpy(stateName[STATE_EOF], "STATE_EOF");

    strcpy(stateName[STATE_START], "STATE_START");
    strcpy(stateName[STATE_STRING], "STATE_STRING");
    strcpy(stateName[STATE_DIGIT], "STATE_DIGIT");

    strcpy(stateName[STATE_ACC_STRING], "STATE_ACC_STRING");
    strcpy(stateName[STATE_ACC_DIGIT], "STATE_ACC_DIGIT");

    strcpy(stateName['!'], "STATE_ACC_Exclamation");
    strcpy(stateName['#'], "STATE_ACC_Fragment");
    strcpy(stateName['$'], "STATE_ACC_Dollar");
    strcpy(stateName['&'], "STATE_ACC_Ampersand");
    strcpy(stateName['/'], "STATE_ACC_Path");
    strcpy(stateName[':'], "STATE_ACC_Detail");
    strcpy(stateName['='], "STATE_ACC_Equal");
    strcpy(stateName['?'], "STATE_ACC_Query");
    strcpy(stateName['@'], "STATE_ACC_UserInfo");
}
```

```
void initTokenName()
{
    for (int i = 0; i <= MAX_TOKEN_VALUE; ++i)
        strcpy(tokenName[i], "TOKEN_UNDEFINED");

    strcpy(tokenName[TOKEN_STRING], "TOKEN_STRING");
    strcpy(tokenName[TOKEN_DIGIT], "TOKEN_DIGIT");

    strcpy(tokenName[TOKEN_HTTP], "TOKEN_HTTP");
    strcpy(tokenName[TOKEN_HTTPS], "TOKEN_HTTPS");
    strcpy(tokenName[TOKEN_FILE], "TOKEN_FILE");
    strcpy(tokenName[TOKEN_MAILTO], "TOKEN_MAILTO");

    strcpy(tokenName['!'], "TOKEN_Exclamation");
    strcpy(tokenName['#'], "TOKEN_Fragment");
    strcpy(tokenName['$'], "TOKEN_Dollar");
    strcpy(tokenName['&'], "TOKEN_Ampersand");
    strcpy(tokenName['/'], "TOKEN_Path");
    strcpy(tokenName[':'], "TOKEN_Detail");
    strcpy(tokenName['='], "TOKEN_Equal");
    strcpy(tokenName['?'], "TOKEN_Query");
    strcpy(tokenName['@'], "TOKEN_UserInfo");
}

char* getTokenNameByTokenNumber(tokenNumber cur){
    return tokenName[cur];
}
```

```
struct Token* makeTokenByState
(struct Token* tokenList, int state, char* value)
{
    if (tokenList == NULL)
        printError(ERROR_LACK_OF_MEMORY,
            "makeTokenByState(struct Token* tokenList, stateNumber state, char* value)");
    else {
        strcpy(tokenList->value, "\0");
        tokenList->number = getTokenNumberByState(state, value);

        // String과 Digit 토큰만 value를 저장해준다.
        switch (tokenList->number)
        {
        case TOKEN_DIGIT:
        case TOKEN_STRING:
            strcpy(tokenList->value, value);
        default: break;
        }
    }
    // 토큰 완성
    return ++tokenList;
}
```

```
stateNumber getState
(stateNumber cur, char input)
{
    stateNumber next = stateTable[cur][input];
    if (next == STATE_ERROR)
        printError(ERROR_UNDEFINED, "getState(stateNumber cur, char input)");
    else return next;
}
```

어휘 분석기 구현 과정 중 느낀 점

- 프로그램 자체의 규모가 커지면서 구현 도중에 프로그램 방향성을 바꾸는 것이 굉장히 어려웠다. 이에 **초기 설계가 매우 중요하다**는 생각이 들었다.
- 뿐만 아니라, 버그가 발생하는 경우에도 **예외 처리**를 하지 않은 함수는 디버깅이 힘들었다. 예외 처리를 위해, 배열 참조 시에도 함수를 활용했다.