Homework 2, 김현학

1. source code

```
// 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;
}
```

```
// DFA.h
#ifndef DFA_H
#define DFA_H

#include <malloc.h>
#include "Token.h"
#include "State.h"

#define BUFFER_SIZE_INPUT (2048)

stateNumber initDFA();
tokenNumber isStringKeyword(char* str);
tokenNumber getTokenNumberByState(stateNumber state, char* value);
struct Token* makeTokenByState(struct Token* tokenList, stateNumber state, char* value);

int runDFA(struct Token* tokenList);

#endif // !DFA_H


// DFA.c
#define _CRT_SECURE_NO_WARNINGS
#include "DFA.h"

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;
}

tokenNumber isStringKeyword(char* str)
{
    if (!strcmp(str, "MAILTO")) return TOKEN_MAILTO;
    else if (!strcmp(str, "FILE"))  return TOKEN_FILE;
    else if (!strcmp(str, "HTTP"))  return TOKEN_HTTP;
    else if (!strcmp(str, "HTTPS")) return TOKEN_HTTPS;
    else return TOKEN_STRING;
}

tokenNumber getTokenNumberByState(stateNumber state, char* value)
{
    switch (state)
    {
    case STATE_ACC_STRING: // ACC_STRING은 Keyword인지 확인해야 한다
        return isStringKeyword(value);
```

```
    case STATE_ACC_DIGIT:
        return TOKEN_DIGIT;

    case STATE_ACC_UserInfo:    case STATE_ACC_Exclamation:
    case STATE_ACC_Detail:      case STATE_ACC_Dollar:
    case STATE_ACC_Path:        case STATE_ACC_Ampersand:
    case STATE_ACC_Query:       case STATE_ACC_Equal:
    case STATE_ACC_Fragment:
        return state;

    default: // Accept 상태가 아니라면
        printError(ERROR_WRONG_PARAMETER, "getTokenNumberByState(NON_ACCEPT_STATE)");
    }
    // 토큰 완성
    return TOKEN_UNDEFINED;
}

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;
}

// 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;
```

```
}
```

```
// State.h

#ifndef STATE_H
#define STATE_H
#define _CRT_SECURE_NO_WARNINGS

#include <string.h>
#include "Error.h"
#include "Char.h"

#define MAX_STATE_VALUE ('@')
#define MAX_LENGTH_STATE_NAME (33)

typedef int stateNumber;
typedef enum ENUM_STATE_NUMBER
{
    STATE_ERROR = -1,
    STATE_EOF = '\0',

    STATE_START,
    STATE_STRING,   //  [ALPHA] *(ALPHA | DIGIT | '-' | '.' | '_' )
    STATE_DIGIT,    // +[DIGIT]

    // --------------
    //  ACCEPT STATE
    // --------------

    // GENERAL
    STATE_ACC_STRING = 10,
    STATE_ACC_DIGIT = 11,

    // DELIM : Value [33, 65)
    STATE_ACC_UserInfo      = '@', // ACC_DELIM_GEN
    STATE_ACC_Detail        = ':', // @ : / ? #
    STATE_ACC_Path          = '/',
    STATE_ACC_Query         = '?',
    STATE_ACC_Fragment      = '#',

    STATE_ACC_Exclamation   = '!', // ACC_DELIM_SUB
    STATE_ACC_Dollar        = '$', // ! $ & =
    STATE_ACC_Ampersand     = '&',
    STATE_ACC_Equal         = '=',

} ENUM_STATE_NUMBER;

void initStateTable();
void initStateName();
int isAcceptState(stateNumber cur);
stateNumber getState(stateNumber cur, char input);
char* getStateName(stateNumber cur);

#endif // !STATE_H


// State.c

#define _CRT_SECURE_NO_WARNINGS
#include "State.h"
#include "Char.h"

// STATE Transition Table
stateNumber stateTable[MAX_STATE_VALUE + 1][MAX_TERMINAL_VALUE + 1] = { STATE_ERROR, };
char stateName[MAX_STATE_VALUE + 1][MAX_LENGTH_STATE_NAME + 1];


void log(int startStateNumber, char inputCharacter, int DestStateNumber) {
    printf("%s:\t\'%c\'->\t%s\n", getStateName(startStateNumber), inputCharacter,
getStateName(DestStateNumber));
}

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");
}

int isAcceptState(stateNumber n) {
    switch (n)
    {
        // GENERAL
    case STATE_ACC_STRING:   case STATE_ACC_DIGIT:
        // DELIM
    case STATE_ACC_UserInfo: case STATE_ACC_Exclamation:
    case STATE_ACC_Detail:   case STATE_ACC_Dollar:
    case STATE_ACC_Path:     case STATE_ACC_Ampersand:
    case STATE_ACC_Query:    case STATE_ACC_Equal:
    case STATE_ACC_Fragment:
        return True;

    default:
        return False;
```

```
    }
}

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;
}
char* getStateName(stateNumber a) {
    return *(stateName + a);
}
```

```
// Token.h

#ifndef TOKEN_H
#define TOKEN_H

#include <string.h>
#include "Error.h"

typedef int tokenNumber;

#define MAX_TOKEN_VALUE ('@')
#define SIZE_TOKEN_BUFFER (256)
#define MAX_LENGTH_TOKEN_NAME (18)

typedef enum ENUM_TOKEN_NUMBER
{
    TOKEN_UNDEFINED = -1,

    // GENERAL FORM
    TOKEN_STRING = 0,   // ( [a, z] | [A, Z] ) *(ALPHA | DIGIT | '-' | '.' | '_' )
    TOKEN_DIGIT = 1,    // [1, 9] *[0, 9]

    // SPECIAL FORM

    // KEYWORD
    TOKEN_HTTP = 10, // SCHEME
    TOKEN_HTTPS,     // Ignore Letter Case
    TOKEN_FILE,      // (Default: Lower case)
    TOKEN_MAILTO,

    // DELIMITER : Value [33, 65)
    TOKEN_UserInfo = '@',   // DELIM_GEN
    TOKEN_Detail = ':', // : / ? # @
    TOKEN_Path = '/',
    TOKEN_Query = '?',
    TOKEN_Fragment = '#',

    TOKEN_Exclamation = '!', // DELIM_SUB
    TOKEN_Dollar = '$', // ! $ & =
    TOKEN_Ampersand = '&',
    TOKEN_Equal = '=',

} ENUM_TOKEN_NUMBER;


struct Token {
    tokenNumber number;
    char value[SIZE_TOKEN_BUFFER];
};

void initTokenName();

char* getTokenNameByTokenNumber(tokenNumber cur);

// STATE Transition Table
tokenNumber tokenTable[MAX_TOKEN_VALUE + 1];
char tokenName[MAX_TOKEN_VALUE + 1][MAX_LENGTH_TOKEN_NAME + 1];

#endif // !TOKEN_H



// Token.c

#define _CRT_SECURE_NO_WARNINGS
#include "Token.h"


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];
}
```

```
// Char.h

#ifndef CHAR_H
#define CHAR_H

#define MIN_TERMINAL_VALUE ('!')
#define MAX_TERMINAL_VALUE ('z')

enum ENUM_TERMINAL_TYPE
{
    Delim, Digit, Letter_Lower, Letter_Upper, Letter
};

char* getTerminalArray(int ENUM_TERMINAL_TYPE);

#endif // !CHAR_H


// Char.c

#define _CRT_SECURE_NO_WARNINGS
#include "Char.h"
#include "Error.h"

char terminalDelim[] = {
    '/', '?', '#', '@', ':',
    '!', '$', '&', '=',
};
char terminalDigit[] = {
    '0','1','2','3','4','5','6','7','8','9'
};
char terminalLetterUpper[] = {
    'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'
};
char terminalLetterLower[] = {
    'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'
};

// terminalString = Digit | Letter | '-' | '.' | '_'
char terminalLetter[] = {
    'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
    'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
    '-', '.', '_',
};

char* getTerminalArray(int type)
{
    switch (type)
    {
    case Delim: return terminalDelim;
    case Digit: return terminalDigit;
    case Letter_Lower: return terminalLetterLower;
    case Letter_Upper: return terminalLetterUpper;
    case Letter: return terminalLetter;
    default:
        printError(ERROR_WRONG_PARAMETER, "getTerminalArray(int type)");
        break;
    }
    return Bad;
}
```

```
// Error.h
```

```h
#ifndef ERROR_H
#define ERROR_H

#include <stdio.h>
#include <stdlib.h>

typedef enum Bool
{
    False = 0,
    True = 1,
} Bool;

typedef enum ErrorCode
{
    Good = 0,
    Bad,
    ERROR_WRONG_PARAMETER,
    ERROR_UNDEFINED,
    ERROR_LACK_OF_MEMORY,
    ERROR_MADE_TOKEN_WITH_NON_ACCEPT_STATE,
} ErrorCode;

void printError(int ErrorCode, char* cur);

#endif // !ERROR_H


// Error.c

#define _CRT_SECURE_NO_WARNINGS
#include "Error.h"

void printError(int ErrorCode, char* cur)
{
    switch (ErrorCode)
    {
    case ERROR_WRONG_PARAMETER:
        printf("ERROR [WRONG_PARAMETER]\n\t%s\n", cur); break;
    case ERROR_UNDEFINED:
        printf("ERROR [UNDEFINED]\n\t%s\n", cur); break;
    case ERROR_LACK_OF_MEMORY:
        printf("ERROR [LACK_OF_MEMORY]\n\t%s\n", cur); break;
    case ERROR_MADE_TOKEN_WITH_NON_ACCEPT_STATE:
        printf("ERROR [ERROR_MADE_TOKEN_WITH_NON_ACCEPT_STATE]\n\t%s\n", cur); break;
    default:
        break;
    }
    exit(ErrorCode);
}
```

2. 실행 화면

```
HTTPS://learn.hansung.ac.kr/06/06mod/board/a.php?id=51&bwid=23#s-40/    <-정상 출력
initStateName....   finished!
initTokenName....   finished!
initStateTable...

Letter: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz-._
STATE_START:    'A'->   STATE_DIGIT
STATE_DIGIT:    'A'->   STATE_DIGIT
STATE_STRING:   'A'->   STATE_STRING

STATE_START:    'B'->   STATE_DIGIT
STATE_DIGIT:    'B'->   STATE_DIGIT
STATE_STRING:   'B'->   STATE_STRING

STATE_START:    'C'->   STATE_DIGIT
STATE_DIGIT:    'C'->   STATE_DIGIT
STATE_STRING:   'C'->   STATE_STRING

STATE_START:    'D'->   STATE_DIGIT
STATE_DIGIT:    'D'->   STATE_DIGIT
STATE_STRING:   'D'->   STATE_STRING

STATE_START:    'E'->   STATE_DIGIT
STATE_DIGIT:    'E'->   STATE_DIGIT
STATE_STRING:   'E'->   STATE_STRING

STATE_START:    'F'->   STATE_DIGIT
STATE_DIGIT:    'F'->   STATE_DIGIT
STATE_STRING:   'F'->   STATE_STRING

STATE_START:    'G'->   STATE_DIGIT
STATE_DIGIT:    'G'->   STATE_DIGIT
```

```
STATE_STRING:   'G'->   STATE_STRING

STATE_START:    'H'->   STATE_DIGIT
STATE_DIGIT:    'H'->   STATE_DIGIT
STATE_STRING:   'H'->   STATE_STRING

STATE_START:    'I'->   STATE_DIGIT
STATE_DIGIT:    'I'->   STATE_DIGIT
STATE_STRING:   'I'->   STATE_STRING

STATE_START:    'J'->   STATE_DIGIT
STATE_DIGIT:    'J'->   STATE_DIGIT
STATE_STRING:   'J'->   STATE_STRING

STATE_START:    'K'->   STATE_DIGIT
STATE_DIGIT:    'K'->   STATE_DIGIT
STATE_STRING:   'K'->   STATE_STRING

STATE_START:    'L'->   STATE_DIGIT
STATE_DIGIT:    'L'->   STATE_DIGIT
STATE_STRING:   'L'->   STATE_STRING

STATE_START:    'M'->   STATE_DIGIT
STATE_DIGIT:    'M'->   STATE_DIGIT
STATE_STRING:   'M'->   STATE_STRING

STATE_START:    'N'->   STATE_DIGIT
STATE_DIGIT:    'N'->   STATE_DIGIT
STATE_STRING:   'N'->   STATE_STRING

STATE_START:    'O'->   STATE_DIGIT
STATE_DIGIT:    'O'->   STATE_DIGIT
STATE_STRING:   'O'->   STATE_STRING

STATE_START:    'P'->   STATE_DIGIT
STATE_DIGIT:    'P'->   STATE_DIGIT
STATE_STRING:   'P'->   STATE_STRING

STATE_START:    'Q'->   STATE_DIGIT
STATE_DIGIT:    'Q'->   STATE_DIGIT
STATE_STRING:   'Q'->   STATE_STRING

STATE_START:    'R'->   STATE_DIGIT
STATE_DIGIT:    'R'->   STATE_DIGIT
STATE_STRING:   'R'->   STATE_STRING

STATE_START:    'S'->   STATE_DIGIT
STATE_DIGIT:    'S'->   STATE_DIGIT
STATE_STRING:   'S'->   STATE_STRING

STATE_START:    'T'->   STATE_DIGIT
STATE_DIGIT:    'T'->   STATE_DIGIT
STATE_STRING:   'T'->   STATE_STRING

STATE_START:    'U'->   STATE_DIGIT
STATE_DIGIT:    'U'->   STATE_DIGIT
STATE_STRING:   'U'->   STATE_STRING

STATE_START:    'V'->   STATE_DIGIT
STATE_DIGIT:    'V'->   STATE_DIGIT
STATE_STRING:   'V'->   STATE_STRING

STATE_START:    'W'->   STATE_DIGIT
STATE_DIGIT:    'W'->   STATE_DIGIT
STATE_STRING:   'W'->   STATE_STRING

STATE_START:    'X'->   STATE_DIGIT
STATE_DIGIT:    'X'->   STATE_DIGIT
STATE_STRING:   'X'->   STATE_STRING

STATE_START:    'Y'->   STATE_DIGIT
STATE_DIGIT:    'Y'->   STATE_DIGIT
STATE_STRING:   'Y'->   STATE_STRING

STATE_START:    'Z'->   STATE_DIGIT
STATE_DIGIT:    'Z'->   STATE_DIGIT
STATE_STRING:   'Z'->   STATE_STRING

STATE_START:    'a'->   STATE_DIGIT
STATE_DIGIT:    'a'->   STATE_DIGIT
STATE_STRING:   'a'->   STATE_STRING

STATE_START:    'b'->   STATE_DIGIT
STATE_DIGIT:    'b'->   STATE_DIGIT
STATE_STRING:   'b'->   STATE_STRING

STATE_START:    'c'->   STATE_DIGIT
```

```
STATE_DIGIT:    'c'->   STATE_DIGIT
STATE_STRING:   'c'->   STATE_STRING

STATE_START:    'd'->   STATE_DIGIT
STATE_DIGIT:    'd'->   STATE_DIGIT
STATE_STRING:   'd'->   STATE_STRING

STATE_START:    'e'->   STATE_DIGIT
STATE_DIGIT:    'e'->   STATE_DIGIT
STATE_STRING:   'e'->   STATE_STRING

STATE_START:    'f'->   STATE_DIGIT
STATE_DIGIT:    'f'->   STATE_DIGIT
STATE_STRING:   'f'->   STATE_STRING

STATE_START:    'g'->   STATE_DIGIT
STATE_DIGIT:    'g'->   STATE_DIGIT
STATE_STRING:   'g'->   STATE_STRING

STATE_START:    'h'->   STATE_DIGIT
STATE_DIGIT:    'h'->   STATE_DIGIT
STATE_STRING:   'h'->   STATE_STRING

STATE_START:    'i'->   STATE_DIGIT
STATE_DIGIT:    'i'->   STATE_DIGIT
STATE_STRING:   'i'->   STATE_STRING

STATE_START:    'j'->   STATE_DIGIT
STATE_DIGIT:    'j'->   STATE_DIGIT
STATE_STRING:   'j'->   STATE_STRING

STATE_START:    'k'->   STATE_DIGIT
STATE_DIGIT:    'k'->   STATE_DIGIT
STATE_STRING:   'k'->   STATE_STRING

STATE_START:    'l'->   STATE_DIGIT
STATE_DIGIT:    'l'->   STATE_DIGIT
STATE_STRING:   'l'->   STATE_STRING

STATE_START:    'm'->   STATE_DIGIT
STATE_DIGIT:    'm'->   STATE_DIGIT
STATE_STRING:   'm'->   STATE_STRING

STATE_START:    'n'->   STATE_DIGIT
STATE_DIGIT:    'n'->   STATE_DIGIT
STATE_STRING:   'n'->   STATE_STRING

STATE_START:    'o'->   STATE_DIGIT
STATE_DIGIT:    'o'->   STATE_DIGIT
STATE_STRING:   'o'->   STATE_STRING

STATE_START:    'p'->   STATE_DIGIT
STATE_DIGIT:    'p'->   STATE_DIGIT
STATE_STRING:   'p'->   STATE_STRING

STATE_START:    'q'->   STATE_DIGIT
STATE_DIGIT:    'q'->   STATE_DIGIT
STATE_STRING:   'q'->   STATE_STRING

STATE_START:    'r'->   STATE_DIGIT
STATE_DIGIT:    'r'->   STATE_DIGIT
STATE_STRING:   'r'->   STATE_STRING

STATE_START:    's'->   STATE_DIGIT
STATE_DIGIT:    's'->   STATE_DIGIT
STATE_STRING:   's'->   STATE_STRING

STATE_START:    't'->   STATE_DIGIT
STATE_DIGIT:    't'->   STATE_DIGIT
STATE_STRING:   't'->   STATE_STRING

STATE_START:    'u'->   STATE_DIGIT
STATE_DIGIT:    'u'->   STATE_DIGIT
STATE_STRING:   'u'->   STATE_STRING

STATE_START:    'v'->   STATE_DIGIT
STATE_DIGIT:    'v'->   STATE_DIGIT
STATE_STRING:   'v'->   STATE_STRING

STATE_START:    'w'->   STATE_DIGIT
STATE_DIGIT:    'w'->   STATE_DIGIT
STATE_STRING:   'w'->   STATE_STRING

STATE_START:    'x'->   STATE_DIGIT
STATE_DIGIT:    'x'->   STATE_DIGIT
STATE_STRING:   'x'->   STATE_STRING
```

```
STATE_START:    'y'->   STATE_DIGIT
STATE_DIGIT:    'y'->   STATE_DIGIT
STATE_STRING:   'y'->   STATE_STRING

STATE_START:    'z'->   STATE_DIGIT
STATE_DIGIT:    'z'->   STATE_DIGIT
STATE_STRING:   'z'->   STATE_STRING

STATE_START:    '-'->   STATE_DIGIT
STATE_DIGIT:    '-'->   STATE_DIGIT
STATE_STRING:   '-'->   STATE_STRING

STATE_START:    '.'->   STATE_DIGIT
STATE_DIGIT:    '.'->   STATE_DIGIT
STATE_STRING:   '.'->   STATE_STRING

STATE_START:    '_'->   STATE_DIGIT
STATE_DIGIT:    '_'->   STATE_DIGIT
STATE_STRING:   '_'->   STATE_STRING

Digit: 0123456789
STATE_START:    '0'->   STATE_DIGIT
STATE_STRING:   '0'->   STATE_STRING
STATE_DIGIT:    '0'->   STATE_DIGIT

STATE_START:    '1'->   STATE_DIGIT
STATE_STRING:   '1'->   STATE_STRING
STATE_DIGIT:    '1'->   STATE_DIGIT

STATE_START:    '2'->   STATE_DIGIT
STATE_STRING:   '2'->   STATE_STRING
STATE_DIGIT:    '2'->   STATE_DIGIT

STATE_START:    '3'->   STATE_DIGIT
STATE_STRING:   '3'->   STATE_STRING
STATE_DIGIT:    '3'->   STATE_DIGIT

STATE_START:    '4'->   STATE_DIGIT
STATE_STRING:   '4'->   STATE_STRING
STATE_DIGIT:    '4'->   STATE_DIGIT

STATE_START:    '5'->   STATE_DIGIT
STATE_STRING:   '5'->   STATE_STRING
STATE_DIGIT:    '5'->   STATE_DIGIT

STATE_START:    '6'->   STATE_DIGIT
STATE_STRING:   '6'->   STATE_STRING
STATE_DIGIT:    '6'->   STATE_DIGIT

STATE_START:    '7'->   STATE_DIGIT
STATE_STRING:   '7'->   STATE_STRING
STATE_DIGIT:    '7'->   STATE_DIGIT

STATE_START:    '8'->   STATE_DIGIT
STATE_STRING:   '8'->   STATE_STRING
STATE_DIGIT:    '8'->   STATE_DIGIT

STATE_START:    '9'->   STATE_DIGIT
STATE_STRING:   '9'->   STATE_STRING
STATE_DIGIT:    '9'->   STATE_DIGIT

Delim: /?#@:!$&=
STATE_STRING:   '/'->   STATE_ACC_STRING
STATE_DIGIT:    '/'->   STATE_ACC_DIGIT
STATE_START:    '/'->   STATE_ACC_Path

STATE_STRING:   '?'->   STATE_ACC_STRING
STATE_DIGIT:    '?'->   STATE_ACC_DIGIT
STATE_START:    '?'->   STATE_ACC_Query

STATE_STRING:   '#'->   STATE_ACC_STRING
STATE_DIGIT:    '#'->   STATE_ACC_DIGIT
STATE_START:    '#'->   STATE_ACC_Fragment

STATE_STRING:   '@'->   STATE_ACC_STRING
STATE_DIGIT:    '@'->   STATE_ACC_DIGIT
STATE_START:    '@'->   STATE_ACC_UserInfo

STATE_STRING:   ':'->   STATE_ACC_STRING
STATE_DIGIT:    ':'->   STATE_ACC_DIGIT
STATE_START:    ':'->   STATE_ACC_Detail

STATE_STRING:   '!'->   STATE_ACC_STRING
STATE_DIGIT:    '!'->   STATE_ACC_DIGIT
STATE_START:    '!'->   STATE_ACC_Exclamation

STATE_STRING:   '$'->   STATE_ACC_STRING
```

```
STATE_DIGIT:    '$'->   STATE_ACC_DIGIT
STATE_START:    '$'->   STATE_ACC_Dollar

STATE_STRING:   '&'->   STATE_ACC_STRING
STATE_DIGIT:    '&'->   STATE_ACC_DIGIT
STATE_START:    '&'->   STATE_ACC_Ampersand

STATE_STRING:   '='->   STATE_ACC_STRING
STATE_DIGIT:    '='->   STATE_ACC_DIGIT
STATE_START:    '='->   STATE_ACC_Equal

finished!
    H            STATE_START >>        STATE_STRING
    T           STATE_STRING >>        STATE_STRING
    T           STATE_STRING >>        STATE_STRING
    P           STATE_STRING >>        STATE_STRING
    S           STATE_STRING >>        STATE_STRING
            STATE_STRING >>     STATE_ACC_STRING
        STATE_ACC_STRING >>          STATE_START

    :            STATE_START >>    STATE_ACC_Detail
        STATE_ACC_Detail >>          STATE_START

    /            STATE_START >>      STATE_ACC_Path
          STATE_ACC_Path >>          STATE_START

    /            STATE_START >>      STATE_ACC_Path
          STATE_ACC_Path >>          STATE_START

    l            STATE_START >>        STATE_STRING
    e           STATE_STRING >>        STATE_STRING
    a           STATE_STRING >>        STATE_STRING
    r           STATE_STRING >>        STATE_STRING
    n           STATE_STRING >>        STATE_STRING
    .           STATE_STRING >>        STATE_STRING
    h           STATE_STRING >>        STATE_STRING
    a           STATE_STRING >>        STATE_STRING
    n           STATE_STRING >>        STATE_STRING
    s           STATE_STRING >>        STATE_STRING
    u           STATE_STRING >>        STATE_STRING
    n           STATE_STRING >>        STATE_STRING
    g           STATE_STRING >>        STATE_STRING
    .           STATE_STRING >>        STATE_STRING
    a           STATE_STRING >>        STATE_STRING
    c           STATE_STRING >>        STATE_STRING
    .           STATE_STRING >>        STATE_STRING
    k           STATE_STRING >>        STATE_STRING
    r           STATE_STRING >>        STATE_STRING
            STATE_STRING >>     STATE_ACC_STRING
        STATE_ACC_STRING >>          STATE_START

    /            STATE_START >>      STATE_ACC_Path
          STATE_ACC_Path >>          STATE_START

    0            STATE_START >>         STATE_DIGIT
    6            STATE_DIGIT >>         STATE_DIGIT
             STATE_DIGIT >>      STATE_ACC_DIGIT
         STATE_ACC_DIGIT >>          STATE_START

    /            STATE_START >>      STATE_ACC_Path
          STATE_ACC_Path >>          STATE_START

    0            STATE_START >>         STATE_DIGIT
    6            STATE_DIGIT >>         STATE_DIGIT
    m            STATE_DIGIT >>        STATE_STRING
    o           STATE_STRING >>        STATE_STRING
    d           STATE_STRING >>        STATE_STRING
            STATE_STRING >>     STATE_ACC_STRING
        STATE_ACC_STRING >>          STATE_START

    /            STATE_START >>      STATE_ACC_Path
          STATE_ACC_Path >>          STATE_START

    b            STATE_START >>        STATE_STRING
    o           STATE_STRING >>        STATE_STRING
    a           STATE_STRING >>        STATE_STRING
    r           STATE_STRING >>        STATE_STRING
    d           STATE_STRING >>        STATE_STRING
            STATE_STRING >>     STATE_ACC_STRING
        STATE_ACC_STRING >>          STATE_START

    /            STATE_START >>      STATE_ACC_Path
          STATE_ACC_Path >>          STATE_START

    a            STATE_START >>        STATE_STRING
    .           STATE_STRING >>        STATE_STRING
    p           STATE_STRING >>        STATE_STRING
```

```
    h           STATE_STRING >>        STATE_STRING
    p           STATE_STRING >>        STATE_STRING
          STATE_STRING >>     STATE_ACC_STRING
      STATE_ACC_STRING >>           STATE_START

    ?            STATE_START >>     STATE_ACC_Query
       STATE_ACC_Query >>           STATE_START

    i            STATE_START >>        STATE_STRING
    d           STATE_STRING >>        STATE_STRING
          STATE_STRING >>     STATE_ACC_STRING
      STATE_ACC_STRING >>           STATE_START

    =            STATE_START >>     STATE_ACC_Equal
       STATE_ACC_Equal >>           STATE_START

    5            STATE_START >>         STATE_DIGIT
    1            STATE_DIGIT >>         STATE_DIGIT
           STATE_DIGIT >>      STATE_ACC_DIGIT
       STATE_ACC_DIGIT >>           STATE_START

    &            STATE_START >> STATE_ACC_Ampersand
     STATE_ACC_Ampersand >>           STATE_START

    b            STATE_START >>        STATE_STRING
    w           STATE_STRING >>        STATE_STRING
    i           STATE_STRING >>        STATE_STRING
    d           STATE_STRING >>        STATE_STRING
          STATE_STRING >>     STATE_ACC_STRING
      STATE_ACC_STRING >>           STATE_START

    =            STATE_START >>     STATE_ACC_Equal
       STATE_ACC_Equal >>           STATE_START

    2            STATE_START >>         STATE_DIGIT
    3            STATE_DIGIT >>         STATE_DIGIT
           STATE_DIGIT >>      STATE_ACC_DIGIT
       STATE_ACC_DIGIT >>           STATE_START

    #            STATE_START >>  STATE_ACC_Fragment
      STATE_ACC_Fragment >>           STATE_START

    s            STATE_START >>        STATE_STRING
    -           STATE_STRING >>        STATE_STRING
    4           STATE_STRING >>        STATE_STRING
    0           STATE_STRING >>        STATE_STRING
          STATE_STRING >>     STATE_ACC_STRING
      STATE_ACC_STRING >>           STATE_START

    /            STATE_START >>      STATE_ACC_Path
         STATE_ACC_Path >>           STATE_START

                     STATE_START >>           STATE_EOF
nToken: 24
TOKEN_HTTPS (11, "")
TOKEN_Detail (58, "")
TOKEN_Path (47, "")
TOKEN_Path (47, "")
TOKEN_STRING (0, "learn.hansung.ac.kr")
TOKEN_Path (47, "")
TOKEN_DIGIT (1, "06")
TOKEN_Path (47, "")
TOKEN_STRING (0, "06mod")
TOKEN_Path (47, "")
TOKEN_STRING (0, "board")
TOKEN_Path (47, "")
TOKEN_STRING (0, "a.php")
TOKEN_Query (63, "")
TOKEN_STRING (0, "id")
TOKEN_Equal (61, "")
TOKEN_DIGIT (1, "51")
TOKEN_Ampersand (38, "")
TOKEN_STRING (0, "bwid")
TOKEN_Equal (61, "")
TOKEN_DIGIT (1, "23")
TOKEN_Fragment (35, "")
TOKEN_STRING (0, "s-40")
TOKEN_Path (47, "")
```

3. sample + result file

- sample file

```
`% ~    <-정의되지 않은 문자가 사용되면 프로그램 종료 (무시하도록 만들 수도 있음)
HTTPS://learn.hansung.ac.kr/06/06mod/ubboard/article.php?id=512393&bwid=240783#s-40/    <-정상 출력
```

- result file

```
    `            STATE_START >>           STATE_EOF
ERROR [WRONG_PARAMETER]
        runDFA(struct Token* tokenList)
```