```
#include "string.h"
#define NULL 0x00
#define MAX_TOKEN_NR 3
#define MAX_KEYWORD_STRING_LTH 10
#define MAX_KEYWORD_NR 3

typedef enum KeywordCode {LD, ST, RST} KeywordCode;

typedef union TokenValue {
  enum KeywordCode eKeyword;
  unsigned int uiNumber;
  char * pcString;
} TokenValue;

typedef enum TokenType {KEYWORD, NUMBER, STRING} TokenType;

typedef struct Token {
  enum TokenType eType;
  union TokenValue uValue;
} Token;

typedef struct Keyword {
  enum KeywordCode eCode;
  char cString[MAX_KEYWORD_STRING_LTH + 1];
} Keyword;

struct Keyword asKeywordList[MAX_KEYWORD_NR] = {
  {RST, "reset"},
  {LD, "load"},
  {ST, "store"}
};

unsigned char ucTokenNr;
struct Token asToken[MAX_TOKEN_NR];

enum State {TOKEN, DELIMITER};
```

```
unsigned char ucFindTokensInString (char *pcString) {

  unsigned char ucCharCounter;
  unsigned char ucCurrentChar;
  enum State eState = DELIMITER;
  ucTokenNr = 0;

  for(ucCharCounter = 0;;ucCharCounter++) {

    ucCurrentChar = pcString[ucCharCounter];

    switch(eState) {
      case DELIMITER:
        if(ucCurrentChar == NULL) {
          return ucTokenNr;
        } else if(ucCurrentChar != ' ') {
          eState = TOKEN;
          asToken[ucTokenNr].uValue.pcString = &pcString[ucCharCounter];
          ucTokenNr++;
        } else {
          eState = DELIMITER;
        }
        break;
      case TOKEN:
        if(ucTokenNr == MAX_TOKEN_NR) {
          return ucTokenNr;
        }  else if(ucCurrentChar == NULL) {
          return ucTokenNr;
        } else if(ucCurrentChar == ' ') {
          eState = DELIMITER;
        } else {
          eState = TOKEN;
        }
        break;
    }
  }
}
```

```
enum Result eStringToKeyword (char pcStr[], enum KeywordCode *peKeywordCode) {

  unsigned char ucKeywordCounter;

  for(ucKeywordCounter = 0;ucKeywordCounter < MAX_KEYWORD_NR;ucKeywordCounter++) {
    if(eCompareString(pcStr, asKeywordList[ucKeywordCounter].cString) == EQUAL) {
      *peKeywordCode = asKeywordList[ucKeywordCounter].eCode;
      return OK;
    }
  }
  return ERROR;
}

void DecodeTokens(void) {

  unsigned char ucTokenCounter;
  struct Token *psCurrentToken;
  unsigned int uiTokenValue;
  enum KeywordCode eTokenCode;

  for(ucTokenCounter = 0; ucTokenCounter < ucTokenNr; ucTokenCounter++) {
    psCurrentToken = &asToken[ucTokenCounter];
    if(eStringToKeyword(psCurrentToken -> uValue.pcString, &eTokenCode) == OK) {
      psCurrentToken -> eType = KEYWORD;
      psCurrentToken -> uValue.eKeyword = eTokenCode;
    } else if(eHexStringToUInt(psCurrentToken -> uValue.pcString, &uiTokenValue) == OK) {
      psCurrentToken -> eType = NUMBER;
      psCurrentToken -> uValue.uiNumber = uiTokenValue;
    } else {
      psCurrentToken -> eType = STRING;
    }
  }
}

void DecodeMsg(char *pcString) {
  ucFindTokensInString(pcString);
  ReplaceCharactersInString(pcString, ' ', NULL);
  DecodeTokens();
}
```