```
#define MAX_TOKEN_NR 3                //maksymalna dopuszczalna ilosc tokenów
#define MAX_KEYWORD_STRING_LTH 10     // maksymalna dlugosc komendy
#define MAX_KEYWORD_NR 3

enum Result { OK, ERROR };
enum KeywordCode { LD, ST, RST };
enum Result { OK, ERROR };
enum TokenType { KEYWORD, NUMBER, STRING };

unsigned char ucTokenNr;     //liczba tokenów w zdekodowanym komunikacie

union TokenValue
{
  enum KeywordCode eKeyword;   // jezeli KEYWORD
  unsigned int uiNumber;       // jezeli NUMBER
  char * pcString;             // jezeli STRING
};

struct Token
{
  enum TokenType eType;        // KEYWORD, NUMBER, STRING
  union TokenValue uValue;     // enum, unsigned int, char*
};

struct Token asToken[MAX_TOKEN_NR]; // wypelniana przez DecodeMsg

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

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

enum eTokenFinderState {TOKEN, DELIMITER};
```

```
unsigned char ucFindTokensInString(char *pcString)
{
  enum eTokenFinderState eFinderState = DELIMITER;
  unsigned char ucCharacterCounter;
  unsigned char ucTokenCounter = 0;
  char cCurrentCharacter;

  for(ucCharacterCounter = 0 ;; ucCharacterCounter++)
  {
    cCurrentCharacter = pcString[ucCharacterCounter];
    switch(eFinderState)
    {
      case TOKEN:
        if (MAX_TOKEN_NR == ucTokenCounter)
        {
          return ucTokenCounter;
        }
        else if ('\0' == cCurrentCharacter)
        {
          return ucTokenCounter;
        }
        else if (' ' != cCurrentCharacter)
        {
          eFinderState = TOKEN;
        }
        else
        {
          eFinderState = DELIMITER;
        }
        break;
      case DELIMITER:
        if ('\0' == cCurrentCharacter)
        {
          return ucTokenCounter;
        }
        else if (' ' == cCurrentCharacter)
        {
          eFinderState = DELIMITER;
        }
        else
        {
          eFinderState = TOKEN;
          asToken[ucTokenCounter].uValue.pcString = pcString + ucCharacterCounter;
          ucTokenCounter++;
        }
        break;
    }
  }
}
```

```
enum Result eSringToKeyword (char pcStr[], enum KeywordCode *peKeywordCode)
{
  unsigned char ucKeyordIterator = 0;

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

void DecodeTokens()
{
  unsigned char ucTokenIndex;

  enum KeywordCode eDecodedKeyword;
  unsigned int uiDecodedNumber;
  struct Token *spCurrentToken;

  for(ucTokenIndex = 0; ucTokenNr > ucTokenIndex; ucTokenIndex++)
  {
    spCurrentToken = &asToken[ucTokenIndex];
    if(OK == eSringToKeyword(spCurrentToken->uValue.pcString, &eDecodedKeyword))
    {
      spCurrentToken->eType = KEYWORD;
      spCurrentToken->uValue.eKeyword = eDecodedKeyword;
    }
    else if (OK == eHexStringToUInt(spCurrentToken->uValue.pcString, &uiDecodedNumber))
    {
      spCurrentToken->eType = NUMBER;
      spCurrentToken->uValue.uiNumber = uiDecodedNumber;
    }
    else
    {
      spCurrentToken->eType = STRING;
    }
  }
}
```

```
void DecodeMsg(char *pcString)
{
  ucTokenNr = ucFindTokensInString(pcString);
  ReplaceCharactersInString(pcString, ' ', '\0');
  DecodeTokens();
}
```