```
#define NULL 0
#define MAX_TOKEN_NR 3
#define MAX_KEYWORD_NR 3
#define MAX_KEYWORD_STRING_LTH 10

typedef enum KeywordCode {LD, ST, RST};

enum Result {ERROR, OK} eResult;

typedef enum TokenType {KEYWORD, NUMBER, STRING};

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

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

struct Token asToken[MAX_TOKEN_NR];

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

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

```
unsigned char ucFindTokensInString(char *pcString)
{
  unsigned char ucCharCounter;
  unsigned char ucCurrentCharacter;
  unsigned char ucNumberOfTokens ;
  enum State { DELIMITER, TOKEN } eState;

  eState = DELIMITER;
  ucNumberOfTokens = 0;
  for ( ucCharCounter = 0; ; ucCharCounter++ )
  {
    ucCurrentCharacter = pcString[ucCharCounter];
    switch ( eState )
    {
      case DELIMITER:
      {
        if ( ucCurrentCharacter == NULL )
        {
          return ucNumberOfTokens;
        }
        else if ( ucNumberOfTokens == MAX_TOKEN_NR )
        {
          return ucNumberOfTokens;
        }
        else if ( ucCurrentCharacter != ' ' )
        {
          eState = TOKEN;
          asToken[ucNumberOfTokens].uValue.pcString = &pcString[ucCharCounter];
          ucNumberOfTokens++;
        }
        else
        {
          eState = DELIMITER;
        }
        break;
      }

      case TOKEN:
      {
        if ( ucCurrentCharacter == NULL )
        {
          return ucNumberOfTokens;
        }
        else if ( ucCurrentCharacter == ' ' )
        {
          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 ucTokenNr;
  struct Token *TokenValue;

  for ( ucTokenNr=0; ucTokenNr < MAX_TOKEN_NR; ucTokenNr++ )
  {
    TokenValue = &asToken[ucTokenNr];

    if ( eStringToKeyword( TokenValue->uValue.pcString, &TokenValue->uValue.eKeyword ) == OK  )
    {
      TokenValue->eType = KEYWORD;
    }
    else if ( eHexStringToUInt ( TokenValue->uValue.pcString, &TokenValue->uValue.uiNumber ) == OK )
    {
      TokenValue->eType = NUMBER;
    }
    else
    {
      TokenValue->eType = STRING;
    }
  }
}

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