Command Decoding in Modula-2

Andersson, Leif

1989

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7413)/1–7/(1989)

Command Decoding in Modula-2

Leif Andersson

Department of Automatic Control
Lund Institute of Technology
January 1989

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
INTERNAL REPORT

Date of issue
1989-01-25

Document Number
CODEN: LUTFD2/(TFRT-7413)/1-7/(1989)

Author(s)
Leif Andersson

Supervisor

Sponsoring organisation

Title and subtitle
Command Decoding in Modula-2

Abstract

Describes how to program command decoding in Modula-2 using the modules LexicalAnalyser and Identifier.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
7

Recipient's notes

Security classification

1. Introduction

This report describes how to program command decoding in Modula-2 using the modules LexicalAnalyser and Identifiers. A simplified version of an Operator's Communication program for a PID regulator is used as an example. The report is oganized as follows. Section 2 describes the commands and parameters accepted by the example program. Section 3 contains the Library Description for the modules LexicalAnalyser and Identifiers, and section 4 contains an annotated listing of the program.

2. The Example Language

The example program is the Operator's Communication Module for a simple PID regulator. It accepts the following commands:

`RUN` starts the regulator
`STOP` stops the regulator
`TSAMP` ⟨number⟩ sets the sampling interval
`PAR` ⟨parameter list⟩ sets one or more parameters
`EXIT` leaves the program

A ⟨parameter list⟩ can contain any number of parameter–value pairs, where the possible parameters are `K`, `TI`, `TD`, `TR` and `N`, and a value is a real number. An example of a valid parameter command is

```
PAR K 2.5 TI 3.8
```

A number of simplifications have been made in order to shorten the code. For example: If the regulator is already running, the command `RUN` should be illegal and give an error message. The converse should be true for the `STOP` command. It should not be possible to change `TSAMP` when the regulator is running.

The only action that is performed by the program is to write the current state to the screen. It is thus possible to see what the result of a command would have been in a real situation.

The `PAR` command is special in that it may contain many parts. If any of these parts is illegal, then the entire `PAR` command is discarded.

3. Library Descriptions

LexicalAnalyser

`DEFINITION MODULE LexicalAnalyser;`

The routines in this module are used to decode a string. The function `LexScan` decodes the next item in the string and returns a value indicating the type of the decoded item. A call to one of the procedures `LexCardinal` through `LexString` will then return the decoded value.

```
EXPORT QUALIFIED
  LexHandle, LexTypes, LexInit, LexInput, LexScan, LexCardinal,
  LexInteger, LexReal, LexIdent, LexDelim, LexString;
```

`TYPE LexHandle;`

A pointer type defined internally in the LexicalAnalyser.

```
TYPE LexTypes =
  (CardLex, CardIntLex, IntLex, RealLex, IdentLex, DelimLex,
   StringLex, EolnLex, EofLex, ErrorLex, RealErrorLex,
   StringErrorLex);
```

The possible results from LexScan. RealErrorLex corresponds to real overflow or underflow. StringError is given when the end of a string is missing.

```
PROCEDURE LexInit(VAR lh: LexHandle);
```

Initializes the internal data structures and returns a handle.

```
PROCEDURE LexInput(lh: LexHandle; s: ARRAY OF CHAR);
```

Makes the string s ready to be decoded.

```
PROCEDURE LexScan(lh: LexHandle): LexTypes;
```

Decodes the next item in the string which is connected with the LexHandle lh. The items must obey the following syntax.

⟨number⟩ :== [+|-]{⟨digit⟩}*[.{⟨digit⟩}*][⟨exponent⟩]

⟨exponent⟩ :== e|E[+|-]{⟨digit⟩}*

⟨digit⟩ :== 0| .. |9

⟨identifier⟩ :== ⟨letter⟩ {⟨letter⟩|⟨digit⟩}*

⟨letter⟩ :== a| .. |z|A| .. |Z

⟨string⟩ :== '{⟨character⟩}*'|"{⟨character⟩}*"

⟨character⟩ :== ⟨all characters defined in the ASCII table⟩

```
PROCEDURE LexCardinal(lh: LexHandle): CARDINAL;
```

Returns the decoded value if the result from LexScan is either CardLex or CardIntLex.

```
PROCEDURE LexInteger(lh: LexHandle): INTEGER;
```

Returns the decoded value if the result from LexScan is either CardIntLex or IntLex.

```
PROCEDURE LexReal(lh: LexHandle): REAL;
```

Returns the decoded value if the result from LexScan is in CardLex .. RealLex.

```
PROCEDURE LexIdent(lh: LexHandle; VAR s: ARRAY OF CHAR);
```

Returns the identifier in s if the result from LexScan is IdentLex. All the letters ('a'..'z') are converted to ('A'..'Z').

```
PROCEDURE LexDelim(lh: LexHandle): CHAR;
```

Returns the delimiter if the result from LexScan is DelimLex.

```
PROCEDURE LexString(lh: LexHandle; VAR s: ARRAY OF CHAR);
```

Returns the string value in s if the result from LexScan is StringLex. The string delimiters are removed. In all other cases LexString returns the part of the input string that constitutes the last decoded item.

```
END LexicalAnalyser.
```

Identifiers

```
DEFINITION MODULE Identifiers;
```
Module to decode identifiers.
```
EXPORT QUALIFIED
  identset, NewIdentSet, BuildIdentSet, SearchIdentSet,
  SearchIdentSetAbbrev;

TYPE identset;

PROCEDURE NewIdentSet(VAR id: identset);
```
Initializes an ident set and returns a reference to it.
```
PROCEDURE BuildIdentSet(id: identset; name: ARRAY OF CHAR;
                        key: CARDINAL);
```
Inserts an identifier in an ident set and assigns a key to it.

`id` The ident set to be used.

`name` The identifier.

`key` The key to be associated with the identifier `name`. The value of key should be `[1..255]` if `SearchIdentSet` will be used and `[2..255]` if SearchIdentSetAbbrev will be used. See these procedures.

```
PROCEDURE SearchIdentSet(id: identset;
                         name: ARRAY OF CHAR): CARDINAL;
```
Searches for an identifier and returns its key if it is found and 0 otherwise.

`id` The ident set to be used.

`name` The identifier

```
PROCEDURE SearchIdentSetAbbrev(id: identset;
                               name: ARRAY OF CHAR): CARDINAL;
```
Searches for an identifier. Any nonambiguous abbreviation of the identifier is acceptable. If the identifier is found, its key is returned. If the abbreviation is ambiguous then 1 is returned and if the identifier is not found then 0 is returned.

`id` The ident set to be used.

`name` The identifier

```
END Identifiers.
```

4. Listing of the Program

```
MODULE OpcomExample;
```
Example program showing the use of the modules `LexicalAnalyser` and `Identifiers`
```
FROM Identifiers IMPORT
  identset, NewIdentSet, BuildIdentSet, SearchIdentSet;

FROM LexicalAnalyser IMPORT
  LexHandle, LexTypes, LexInit, LexInput, LexScan, LexCardinal,
  LexInteger, LexReal, LexIdent, LexDelim, LexString;

FROM ConvReal IMPORT RealToString;
```

```
FROM Terminal IMPORT ReadString, Write, WriteString, WriteLn;

VAR running: BOOLEAN;  True if the (imagined) regulator is running.
VAR leaving: BOOLEAN;  This variable is true if the EXIT command has been
                       given, false otherwise
VAR tsamp: REAL;       Sampling interval of the (imagined) regulator
VAR lh: LexHandle;     The handle returned from LexicalAnalyser

TYPE ParRec = RECORD
  This record type contains the controller parameters.
  k: REAL;  regulator gain
  ti: REAL; integrator time
  td: REAL; derivative time
  tr: REAL; input constant
  n: REAL;  derivation filter factor
END (* ParRec *);

VAR ParA, ParB: ParRec;
  One of these variables (ParA) receives data from the PAR command. If all
  is OK, then the entire record is copied to ParB, which is the one actually
  used by the (imagined) regulator.

TYPE string = ARRAY [0..79] OF CHAR;
  All character arrays of the module are of this type.

VAR  instring: string; This variable receives all input from the user
```

The following two local modules, CommandModule and ParamModule serve the purpose of keeping together all handling of an identset and the enumeration type connected with it. Each of the modules also contain a function procedure which is really just a type converter for SearchIdentSet.

```
MODULE CommandModule;
  IMPORT identset, NewIdentSet, SearchIdentSet, BuildIdentSet;
  EXPORT Commands, SearchCommands;
  VAR commandset: identset;
  TYPE Commands = (nocommand, tsampx, runx, stopx, parx, exitx);
  PROCEDURE SearchCommands(id: ARRAY OF CHAR): Commands;
  BEGIN
    RETURN VAL(Commands,SearchIdentSet(commandset,id));
  END SearchCommands;
BEGIN
  NewIdentSet(commandset);
  BuildIdentSet(commandset, "TSAMP", ORD(tsampx));
  BuildIdentSet(commandset, "RUN", ORD(runx));
  BuildIdentSet(commandset, "STOP", ORD(stopx));
  BuildIdentSet(commandset, "PAR", ORD(parx));
  BuildIdentSet(commandset, "EXIT", ORD(exitx));
END CommandModule;

MODULE ParModule;
  IMPORT identset, NewIdentSet, SearchIdentSet, BuildIdentSet;
  EXPORT Parameters, SearchParams;
  VAR paramset: identset;
```

```
  TYPE Parameters = (noparam, kx, tix, tdx, trx, nx);
  PROCEDURE SearchParams(id: ARRAY OF CHAR): Parameters;
  BEGIN
    RETURN VAL(Parameters,SearchIdentSet(paramset,id));
  END SearchParams;
BEGIN
  NewIdentSet(paramset);
  BuildIdentSet(paramset, "K", ORD(kx));
  BuildIdentSet(paramset, "TI", ORD(tix));
  BuildIdentSet(paramset, "TD", ORD(tdx));
  BuildIdentSet(paramset, "TR", ORD(trx));
  BuildIdentSet(paramset, "N", ORD(nx));
END ParModule;
```

```
PROCEDURE WriteStatus;
```

This procedure performs the only "action" of the module: it writes the status of the (imagined) regulator and the values of the parameters.

```
VAR s: string;
BEGIN
  WITH ParB DO
    IF running THEN
      WriteString("Running");
    ELSE
      WriteString("Stopped");
    END;
    WriteLn;
    WriteString("tsamp = ");
    RealToString(tsamp,s,6); WriteString(s);
    RealToString(k,s,6);
    WriteString("    k = "); WriteString(s);
    RealToString(n,s,6);
    WriteString("   n = "); WriteString(s); WriteLn;
    RealToString(ti,s,6);
    WriteString("ti = "); WriteString(s);
    RealToString(td,s,6);
    WriteString("   td = "); WriteString(s);
    RealToString(tr,s,6);
    WriteString("   tr = "); WriteString(s); WriteLn;
  END (* with *);
END WriteStatus;
```

```
PROCEDURE GetCommand;
```

Decodes a command from the input string and dispatches to other procedures if applicable.

```
VAR
  lt: LexTypes;
  id: string;
  done: BOOLEAN;
BEGIN
  lt:=LexScan(lh);
  IF lt = EolnLex THEN RETURN END;
  IF lt <> IdentLex THEN
    WriteString("Bad command."); WriteLn;
```

```
    RETURN;
  END;
  LexIdent(lh,id);
  CASE SearchCommands(id) OF
    nocommand:
      WriteString("Command not found"); WriteLn;|
    tsampx: Getreal(tsamp,done);|
    runx: running := TRUE;|
    stopx:  running := FALSE;|
    parx: GetParameters;|
    exitx: leaving := TRUE;
  END case ;
END GetCommand;

PROCEDURE Getreal(VAR result: REAL; VAR done: BOOLEAN);
  Decodes a real number from the input string. If all is OK then the number
  is returned in result and done=TRUE else done=FALSE
VAR lt: LexTypes;
BEGIN
  lt := LexScan(lh);
  IF lt <= RealLex THEN
    result := LexReal(lh);
    done := TRUE;
  ELSE
    WriteString("Real number expected."); WriteLn;
    done := FALSE;
  END;
END Getreal;

PROCEDURE GetParameters;
  Reads the parameters into ParA, and if all is OK, transfers them to ParB
VAR done: BOOLEAN; id: string; lt: LexTypes;
BEGIN
  done := FALSE;
  WITH ParA DO
    LOOP
      lt:=LexScan(lh);
      IF lt = EolnLex THEN EXIT END;
      IF lt <> IdentLex THEN
        WriteString("Parameter name expected.");
        EXIT;
      END;
      LexIdent(lh,id);
      CASE SearchParams(id) OF
      noparam:
        WriteString("Unknown parameter name ");
        WriteString(id); WriteLn;
        EXIT;|
      kx: Getreal(k,done);|
      tix: Getreal(ti,done);|
      tdx: Getreal(td,done);|
      trx: Getreal(tr,done);|
```

```
      nx: Getreal(n,done);
      END case ;
      IF NOT done THEN EXIT END;
    END (* loop *);
  END (* with *);
  IF done THEN ParB := ParA; END;
END GetParameters;

BEGIN
  Some initialization code.
  LexInit(lh);
  tsamp := 1.0;
  WITH ParA DO
    k := 1.0; ti:= 1.0; td := 1.0;
    tr := 1.0; n := 10.0;
  END;
  ParB := ParA;
  leaving := FALSE; running:=FALSE;
    This is the main loop
  REPEAT
    WriteStatus;
    Write('>'); ReadString(instring); WriteLn;
    LexInput(lh, instring);
    GetCommand;
  UNTIL leaving;
END OpcomExample.
```

5. References

LOGITECH (1987): *Logitech Modula-2 User's Manual*, Logitech, Fremont, CA.