T800 Tool - Anleitung

-17.6.1990-

Inhalt

A. Vorwort

Diese Anleitung geht davon aus, dass der Benutzer mit dem Oberon System von Ceres vertraut ist.

Die auf Ceres entwickelte Sprache Oberon ist heute auch auf anderen Maschinen wie IBM-PC/AT und PS/2, Macintosh II und SparcStation I verfügbar. Im Rahmen dieser Diplomarbeit wurde der Oberon Compiler weiter auf den Transputer T800 portiert. Diese Anleitung soll das Testen und die Weiterentwickung dieser Implementation erleichtern.

Der T800 Tool ist ein kleines Entwicklungssystem, das eine einfache Testumgebung für den T800 Oberon Cross-Compiler TOP2 bietet. Mit TOP2 wurde eine vollständige Implementation von Oberon realisiert (gelbes Bericht Nr.111) und bis auf die Type-Tests/Guards getestet.

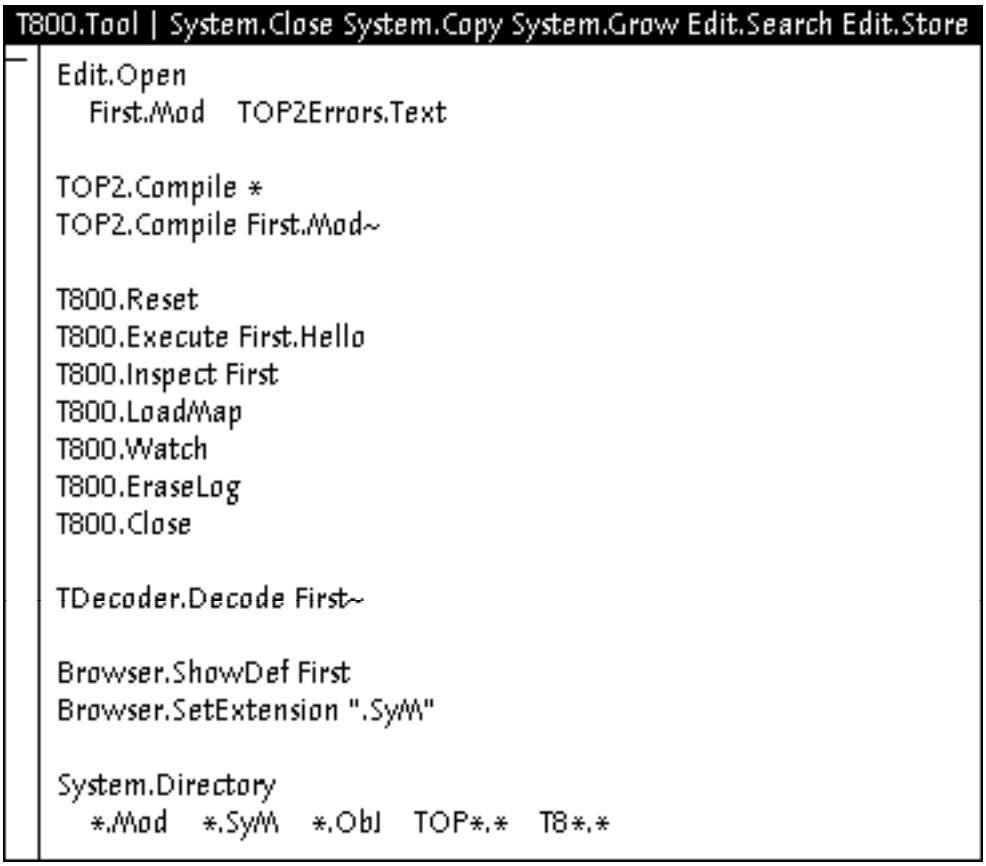
B. Installation

Das T800 Tool wird einfach durch das Kopieren der begleitenden Floppy-Disk installiert.

Das Runtime System ausgenommen funktioniert die hier beschriebene Software auf allen Ceres-Maschinen, deren Oberon-Release der Version vom Juni 1990 entspricht. Kompilierte Oberon Programme werden auf einer mit der Transputer-karte ausgestatteten Ceres ausgeführt.

C. Tools

Das T800 Tool kann mit `System.Open T800.Tool` geöffnet werden; es sieht etwa so aus:



Die Testumgebung bildet die Umgebung des Oberon Systems nach, nämlich:

	Ceres	T800

Viewers :	`System.Tool`
	`System.Log`
	`System.Trap`
		`T800.Tool`
		`T800.Log`
		`T800.Trap`

Tools :	`NOP2.Compile`
	`Decoder.Decode`
	`System.LoadMap`
	`EraseTool.EraseLog`
	`Browser.ShowDef`
		`TOP2.Compile`
		`TDecoder.Decode`
		`T800.LoadMap`
		`T800.EraseLog`
		`Browser.ShowDef`

Kommando-Interpreter :	"Ceres"-Kommandos werden vom mit jedem Viewer assozierten Text-Handler interpretiert	"T800"-Kommandos werden vom "Ceres"-Kommando `T800.Execute` interpretiert

Tools	Kommandos	Parameter	Beispiele
TOP2	Compile	{name}~	Compile Test.Mod~
	Compile	*	
T800	Reset	-	
	Execute	name	Execute Test.Do
	Inspect	name	Inspect Test
	LoadMap	-	
	Watch	-	
	EraseLog	-	
	Close	-	
TDecoder	Decode	{name}~	Decode Test~
TDec	Prefix	{number}~	Prefix 10 0FFH -2~
	Generate	{name}~	Generate T800~
	Decode	{name}~	Decode T800~
Browser	ShowDef	{name}~	ShowDef Terminal~
	SetExtension	string	SetExtension ".SyM"

TOP2

Der Oberon Cross-Compiler TOP2 besteht aus zehn Object-Files: `TOP2`, `TOPV`, `TOPB`, `TOPT`, `TOPS`, `TOPV`, `TOPCa`, `TOPC`, `TOPL` und `TOPM`. Mit `System.Directory TOP*.*` können die vorhandenen Files aufgelistet werden.

`TOP2.Compile`

Die Compiler Optionen entsprechen denen von Ceres und lassen sich frei kombinieren:

```
/x   no index check
/v   overflow check
/r   range check
/t   type check
/s   new symbol file
```

Fehler werden im Viewer `System.Log` ausgegeben.

Als Resultat einer erfolgreichen Kompilation werden von TOP2 zwei Files generiert, nämlich ein Symbol-File mit Suffix "`.SyM`" und ein Object-File mit Suffix "`.ObJ`".

T800

Das T800-Runtime System T800 besteht aus vier Object-Files, nämlich `T800`, `T8Interface`, `T8Modules` und `T8Base` und aus einem Boot-File für das Transputerboard, mit Namen `T800.Boot`. Mit `System.Directory T8*.*` können die vorhandenen Files aufgelistet werden.

`T800.Reset`
Das T800-Runtime System wird reinitialisiert.
Wenn offen, wird der Viewer `T800.Log` geschlossen.

`T800.Execute`
Der Kommando-Interpreter wird aufgerufen (z.B. `T800.Execute First.Hello`).
Fehlermeldungen werden im Viewer `System.Log` ausgegeben.
Wenn nicht vorhanden, wird der Viewer `T800.Log` im User Track geöffnet.

`T800.Inspect`
Ein Memory-Dump der globalen Daten wird im Viewer `System.Log` ausgegeben.
Nach jeder Wortadresse wird der Speicherinhalt in Hexadezimaler und Dezimaler Form und als reelle Zahl angegeben.

`T800.LoadMap`
Die geladenen Moduln werden mit Startadresse, Codegrösse (in Bytes) und Referenz-count aufgelistet (ähnlich wie `System.LoadMap`).

`T800.Watch`
Die Anzahl Bytes, die auf dem *Heap* alloziert sind, wird im Viewer `System.Log` ausgegeben.

`T800.EraseLog`
Der Viewer `T800.Log` wird gelöscht (gleich wie `EraseTool.EraseLog`).

`T800.Close`
Der Viewer `T800.Log` wird geschlossen.

> **Achtung !**
> `System.Free` soll nicht mit "Runtime-Moduln" benutzt werden solange der Viewer `T800.Log` offen ist.

Laufzeitfehler werden im Viewer `T800.Trap` ausgegeben.

Dieser wird im System Track durch das Auftreten eines Fehlers geöffnet.

Folgende Informationen sind aus `T800.Trap` zu entnehmen :

- Die Fehlernummer, deren Interpretation in der Fehlerliste zu finden ist.
- Der FP-Wert (*Frame-Pointer*).
- Der PC-Wert (*Program Counter*).
- Der Name und die Codegrösse des betroffenen Moduls (falls möglich).

Wenn vorhanden verweist die in eckigen Klammern angegebene Adresse auf den PC-Wert in dem entsprechenden dekodierten Text-File.

TDecoder

Der T800-Decoder besteht aus einem einzigen Object-File, nämlich `TDecoder`. Für eine saubere Ausgabe wird der Font-File `Courier10.Scn.Fnt` empfohlen (nicht proportionale Schriftart).

`TDecoder.Decode`
Als Parameter wird eine Liste von Modulnamen erwartet; falls nicht vorhanden wird der Suffix "`.ObJ`" als *default* angenommen. Das dekodierte Code wird in einem Viewer mit Namen "`<modulename>.DeC`" ausgegeben und kann mit `Edit.Store` gespeichert werden. Der Output für ein kleines Programm ist im nächsten Abschnitt als Beispiel angegeben.

TDec

Mit TDec lässt sich das Boot-File `T800.Boot` auf schnelle Art ändern; Modifikationen sind aber mit Vorsicht vorzunehmen.

`TDec.Prefix`
Als Parameter wird eine Liste von (hexa-)dezimalen Zahlen erwartet. Diese werden in ihrer "Prefix-Form" - so werden Operanden bei Transputer Instruktionen erwartet - im Viewer `System.Log` ausgegeben.

`TDec.Generate`
In der Parameterliste werden Namen von "Code-Files" erwartet; falls nicht vorhanden wird der Suffix "`.HEX`" als *default* angenommen. Ein Code-File enthält eine Sequenz von Bytes, die als positive (hexa-)dezimale Zahlen geschrieben werden. Die erste Zahl wird als Codelänge, die übrigen als Codenummern für den Transputer interpretiert. Eine Datei mit Suffix ".OBJ" wird auf der Festplatte gespeichert.

`TDec.Decode`
Der dekodierte Boot-File wird in einem Viewer mit Namen "`<filename>.DEC`" ausgegeben und kann mit `Edit.Store` gespeichert werden.

Browser

`Browser.ShowDef`
Die Schnittstelle des als Parameter angegebenen Moduls wird in einem Viewer mit Namen "`<modulename>.Def`" ausgegeben.

`Browser.SetExtension`
Symbol-Files, die mit Ceres Compiler (NOP2 oder Compiler) oder mit TOP2 erzeugt werden, unterscheiden sich nur durch ihren Suffix, nämlich "`.Sym`" beziehungsweise "`.SyM`". Mit dem Kommando `Browser.SetExtension` kann dieser entsprechend gesetzt werden.

D. Beispiel

Folgendes Programm soll editiert, compiliert, ausgeführt und dekodiert werden.

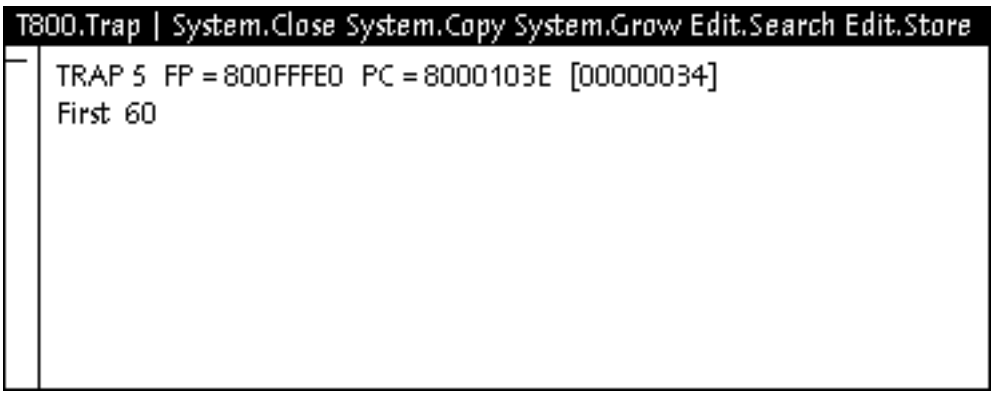
```
MODULE First;

   IMPORT Terminal;

   PROCEDURE Hello*;
   BEGIN
     Terminal.WriteStr("Hello");
     Terminal.WriteLn
   END Hello;

   PROCEDURE Buggy*;
     VAR i, j: INTEGER;
   BEGIN
     i:= 0; j:= 10 DIV i    (* division by zero *)
   END Buggy;

END First.
```

Mit `Edit.Open First.Mod` und `Edit.Store` wird ein Text-File mit Namen `First.Mod` auf der Festplatte gespeichert.

Mit `TOP2.Compile First.Mod~` wird ein Symbol-File mit Namen `First.SyM` und ein Object-File mit Namen `First.ObJ` erzeugt.

Mit `T800.Execute First.Hello` wird das Programm ausgeführt und die Meldung "`Hello`" im Viewer `T800.Log` (beliebig oft) ausgegeben.

Mit `T800.Execute First.Buggy` wird im Viewer `T800.Trap` ein Laufzeitfehler gemeldet. In der Fehlerliste entspricht die Fehlernummer 5 einer Division durch Null.



Mit `TDecoder.Decode First~` wird das dekodierte Code im Viever `First.DeC` ausgegeben. Mit der in eckigen Klammern angegebenen Adresse (hier [`00000034`]) vom Viewer `T800.Trap` kann im Output die betroffene Instruktion nachgegesucht werden.

```
TDecoder  SM 26.4.90
[10.08.90 09:15:31]

HEADER
   Name, Key    = First, 1667251004
   Constant size= 8
   Data size    = 0
   Code size    = 60

COMMANDS
   Name, Entry  = Hello, 0005
   Name, Entry  = Buggy, 002C

IMPORT
   Name, Key    = Terminal, -1012857612

CODE
   0000:  D0H              stl      0 (1)
   0001:  22H   F0H        ret   (6)

PROCEDURE Hello
   0003:  60H   BCH        ajw     -1 (2)
   0005:  D0H              stl      0 (1)
   0006:  60H   BEH        ajw     -2 (2)
   0008:  41H              ldc      1 (1)
   0009:  21H   FBH        ldpi  (3)
   000B:  60H   5BH        ldnlp   -5 (2)
   000D:  D0H              stl      0 (1)
   000E:  46H              ldc      6 (1)
   000F:  D1H              stl      1 (1)

   usw...

   0027:  B1H              ajw      1 (1)
   0028:  22H   F0H        ret   (6)

PROCEDURE Buggy
   002A:  60H   BCH        ajw     -4 (2)
   002C:  D0H              stl      0 (1)
   002D:  60H   BEH        ajw     -2 (2)
   002F:  40H              ldc      0 (1)
   0030:  D0H              stl      0 (1)
   0031:  4AH              ldc     10 (1)
   0032:  70H              ldl      0 (2)
   0033:  22H   FCH        div   (40)
   0035:  D1H              stl      1 (1)
   0036:  B2H              ajw      2 (1)
   0007:  22H   F0H        ret   (6)
   0039:  20H   20H   20H
```

E. Library Module

Zur Zeit stehen die Moduln `Terminal`, `Files`, `Math`, `MathL` und `Clock` zur Verfügung. Am Ende dieses Abschnitts wird der Benutzer auf besondere Merkmale aufmerksam gemacht.

```
MODULE Terminal;

   PROCEDURE Write(ch: CHAR);
   PROCEDURE WriteByteHex(x: BYTE);
   PROCEDURE WriteHex(x: LONGINT);
   PROCEDURE WriteInt(x, n: LONGINT);
   PROCEDURE WriteLn;
   PROCEDURE WriteLReal(x: LONGREAL; n: INTEGER);
   PROCEDURE WriteLRealHex(x: LONGREAL);
   PROCEDURE WriteReal(x: REAL; n: INTEGER);
   PROCEDURE WriteRealHex(x: REAL);
   PROCEDURE WriteStr(s: ARRAY OF CHAR);
(* press carriage return to end input *)
   PROCEDURE Read(VAR ch: CHAR);
   PROCEDURE ReadInt(VAR x: LONGINT);
   PROCEDURE ReadReal(VAR x: REAL);
   PROCEDURE ReadLReal(VAR x: LONGREAL);
   PROCEDURE ReadStr(VAR s: ARRAY OF CHAR);

END Terminal.


MODULE Files;

   TYPE
     File  = POINTER TO Handle;
     Handle = RECORD END;
     Rider = RECORD res: INTEGER; eof: BOOLEAN END;

   PROCEDURE Base(VAR r: Rider): File;
   PROCEDURE Close(f: File);
   PROCEDURE Delete(name: ARRAY OF CHAR);
   PROCEDURE GetDate(f: File; VAR t, d: LONGINT);
   PROCEDURE Length(f: File): LONGINT;
   PROCEDURE New(name: ARRAY OF CHAR): File;
   PROCEDURE Old(name: ARRAY OF CHAR): File;
   PROCEDURE Pos(VAR r: Rider): LONGINT;
   PROCEDURE Purge(f: File);
   PROCEDURE Read(VAR r: Rider; VAR x: BYTE);
   PROCEDURE ReadBytes(VAR r: Rider; VAR x: ARRAY OF BYTE; n: LONGINT);
   PROCEDURE Register(f: File);
   PROCEDURE Rename(old, new: ARRAY OF CHAR; VAR res: INTEGER);
   PROCEDURE Set(VAR r: Rider; f: File; pos: LONGINT);
   PROCEDURE Write(VAR r: Rider; x: BYTE);
   PROCEDURE WriteBytes(VAR r: Rider; VAR x: ARRAY OF BYTE;
n:LONGINT);

END Files;
```

```
MODULE Math;

   CONST
     e = 2.18281828E+00;
     pi = 3.1415928E+00;

   PROCEDURE ln     (x: REAL): REAL;
   PROCEDURE exp    (x: REAL): REAL;
   PROCEDURE sin    (x: REAL): REAL;
   PROCEDURE cos    (x: REAL): REAL;
   PROCEDURE arctan (x: REAL): REAL;
   PROCEDURE sqrt   (x: REAL): REAL;

END Math;


MODULE MathL;

   CONST
     e = 2.1828182845905D+000;
     pi = 3.14159265358979D+000;

   PROCEDURE ln     (x: LONGREAL): LONGREAL;
   PROCEDURE exp    (x: LONGREAL): LONGREAL;
   PROCEDURE sin    (x: LONGREAL): LONGREAL;
   PROCEDURE cos    (x: LONGREAL): LONGREAL;
   PROCEDURE arctan (x: LONGREAL): LONGREAL;
   PROCEDURE sqrt   (x: LONGREAL): LONGREAL;

END MathL;


MODULE Clock;

   CONST
     oneSec = 15625;
     oneMin = 60 * oneSec;

   PROCEDURE Time (): LONGINT;
   PROCEDURE Delay (time: LONGINT);

END Clock.
```

Merkmale

1) `Terminal`: Für die *read* -Operationen wird die Benutzereingabe im Viewer `T800.Log` erwartet; eine Eingabe wird durch *carriage return* beendet.
2) `Files`: Die Anzahl geöffneter Files ist gegenwärtig auf vier beschränkt. Durch die Operationen `Close(f)` und `Purge(f)` werden die mit `f` assoziierten Riders geschlossen.
3) `Math/MathL`: Die Funktionen `ln`, `exp`, `sin`, `cos` und `arctan` sind noch nicht implementiert und erzeugen einen Trap 17 (Funktion ohne **`RETURN`**).

F. Modul SYSTEM

Der Modul `SYSTEM` stellt einige sogenannte *low-level* Operationen zur Verfügung, die zu dieser Implementation von Oberon spezifisch sind. Bei den aufgelisteten Funktionen und Prozeduren steht `v` für eine Variable, `a`, `n` und `x` für Ausdrücke, und `T` für einen Typ.

Funktionen

Name	Typ der Argumenten	Funktionstyp	Beschreibung
ADR(v)	v: beliebiger Typ	**LONGINT**	Adresse der Variable v
BIT(a,n)	a: **LONGINT** n: Integer-Typ	**BOOLEAN**	n IN Mem[a]
LSH(x,n)	x, n: Integer-Typ	**LONGINT**	Logische Shift
ROT(x,n)	x: Integer-Typ	Typ von x	Rotation
VAL(T,x)	T, x: beliebiger Typ	T	x interpretiert als von Typ T

Prozeduren

Name	Typ der Argumenten	Beschreibung
GET(a,v)	a: **LONGINT**, v: beliebiger Basistyp	v := Mem[a]
GETREG(n,v)	n: Integer-Konstante, v: beliebiger Basistyp	v := Reg n (n = 0)
PUT(a, x)	a: **LONGINT**, x: beliebiger Basistyp	Mem[a] := v
PUTREG(n,x)	n: Integer-Konstante, x: beliebiger Basistyp	Reg n := v (n = 0)
MOVE(v0,v1,n)	v0, v1: beliebiger Typ, n: Integer-Typ (n > 0)	Zuweist n Bytes von v0 zu v1
NEW(v, n)	v: beliebiger **POINTER**-Typ, n: Integer-Typ	Alloziert n Bytes im Speicher und gibt Adresse in v zurück

Die Funktion **SIZE**(T) gibt die für den Typ `T` allozierte Anzahl Bytes zurück. In dieser Implementation von Oberon werden die Basistypen wie folgt intern dargestellt:

Datentyp	interne Darstellung
SHORTINT	1 Byte (mit Vorzeichen)
CHAR, **BYTE**	1 Byte
BOOLEAN	1 Byte (**FALSE** = 0, **TRUE** = 1)
INTEGER, **LONGINT**	4 Bytes (mit Vorzeichen)
SET	4 Bytes ({0} = 1)
POINTER-Typ	4 Bytes (**NIL** = 0)
Prozedur-Typ	4 Bytes
REAL	4 Bytes IEEE-Format (754-1985)
LONGREAL	8 Bytes IEEE-Format (754-1985)

G. Fehlerliste

Der Text-File `TOP2Errors.Text` enthält die Liste der von TOP2 erzeugten Fehler. Bis auf die Beschreibung der Laufzeitfehler (Teil 3) ist diese Liste mit der von Ceres identisch.

N. Wirth / 20.6.87 / RC 7.2.90 / SM 17.6.90 (IMS T800 Oberon Cross-Compiler)

1. Incorrect use of language Oberon

0 undeclared identifier
1 multiply defined identifier
2 illegal character in number
3 illegal character in string
4 identifier does not match procedure name
5 comment not closed
6
7
8
9 "=" expected
10 identifier expected
11
12 type definition starts with incorrect symbol
13 factor starts with incorrect symbol
14 statement starts with incorrect symbol
15 declaration followed by incorrect symbol
16 MODULE expected
17 number expected
18 "." missing
19 "," missing
20 ":" missing
21
22 ")" missing
23 "]" missing
24 "}" missing
25 OF missing
26 THEN missing
27 DO missing
28 TO missing
29 "(" missing
30
31
32
33 ":=" missing
34 "," or OF expected
35
36
37 identifier expected
38 ";" missing
39

40 END missing
41
42
43 UNTIL missing
44
45 EXIT not within loop statement
46
47 illegally marked identifier
48 unsatisfied forward reference
49 recursive import not allowed
50 expression should be constant
51 constant not an integer
52 identifier does not denote a type
53 identifier does not denote a record type
54 result type of procedure is not a basic type
55 procedure call of a function
56 assignment to non-variable
57 pointer not bound to record or array type
58 recursive type definition
59 illegal open array parameter
60 wrong type of case label
61 inadmissible type of case label
62 case label defined more than once
63 index out of bounds
64 more actual than formal parameters
65 fewer actual than formal parameters
66 element types of actual array and formal open array differ
67 actual parameter corresponding to open array is not an array
68
69 parameter must be an integer constant
70
71
72
73 procedure must have level 0
74
75
76
77 object is not a record
78 dereferenced object is not a variable
79 indexed object is not a variable
80 index expression is not an integer
81 index out of specified bounds
82 indexed variable is not an array
83 undefined record field
84 dereferenced variable is not a pointer
85 guard or test type is not an extension of variable type
86 guard or testtype is not a pointer
87 guarded or tested variable is neither a pointer nor a VAR-parameter record
88
89
90

91
92 operand of IN not an integer, or not a set
93 set element type is not an integer
94 operand of & is not of type BOOLEAN
95 operand of OR is not of type BOOLEAN
96 operand not applicable to (unary) +
97 operand not applicable to (unary) -
98 operand of ~ is not of type BOOLEAN
99
100 incompatible operands of dyadic operator
101 operand type inapplicable to *
102 operand type inapplicable to /
103 operand type inapplicable to DIV
104 operand type inapplicable to MOD
105 operand type inapplicable to +
106 operand type inapplicable to -
107 operand type inapplicable to = or #
108 operand type inapplicable to relation
109
110 operand is not a type
111 operand inapplicable to (this) function
112 operand is not a variable
113 incompatible assignment
114 string too long
115 parameter discrepancy between type (or name) of variable (or forward procedure)
and this procedure
116 type of variable (or forward procedure) has more parameters than this procedure
117 type of variable (or forward procedure) has fewer parameters than this procedure
118 procedure result type of variable (or of forward declaration) differs from result type
of this procedure
119 assigned procedure is not global
120 type of expression following IF, WHILE, or UNTIL is not BOOLEAN
121 called object is not a procedure (or is an interrupt procedure)
122 actual VAR-parameter is not a variable
123 type of actual parameter is not identical with that of formal VAR-parameter
124 type of result expression differs from that of procedure
125 type of case expression is neither INTEGER nor CHAR
126 this expression cannot be a type or a procedure
127 illegal use of object
128
129 unsatisfied forward procedure
130 WITH clause does not specify a variable
131 LEN not applied to array
132 dimension in LEN too large or negative
133 procedure declaration don't match forward declaration

150 key inconsistency of imported module

151 incorrect symbol file
152 symbol file of imported module not found
153 object or symbol file not opened (disk full?)
154
155 generation of new symbol file not allowed

2. Limitations of implementation

200 not yet implemented
201 lower bound of set range greater than higher bound
202 set element greater than MAX(SET) or less than 0
203 number too large
204 product too large
205 division by zero
206 sum too large
207 difference too large
208 overflow in arithmetic shift
209 case range too large
210 code too long
211 jump distance too large
212
213 too many cases in case statement
214 too many exit statements
215 not enough registers: simplify expression
216 not enough floating-point registers: simplify expression
217 parameter must be an integer constant
218 illegal value of parameter (20 <= p < 256)
219 illegal value of parameter (0 <= p < 16)
220 illegal value of parameter
221
222 too many pointers (either global, or in record)
223 too many record types
224 too many pointer types
225 address of pointer variable too large (move forward in text)
226 too many exported procedures (> 40)
227 too many imported modules
228 too many exported structures
229 too many nested records for import
230 too many constants (strings) in module
231 too many link table entries (external procedures)
232 too many commands in module
233 record extension hierarchy too high
240 identifier too long
241 string too long
242
255 register not released (compiler error)

3. Run-time Trap Numbers

1 Out of heap space
2 NIL-reference
3
4
5 Division by zero
6 Invalid index
7 Range error in conversion
8 Unknown error
9
10
11
12
13 Integer overflow
14 Floating-point overflow
15

16 Invalid case in CASE statement
17 Function procedure without RETURN statement
18 Type guard check
19 Implied type guard check in record assignment

20 Disk error (unreadable sector)
21 Illegal Sector address
22 Break from keyboard
23 File too long (> 2.5 MB)
24 Disk full
25 Missing file for delayed loading
26
27 Illegal function argument (Math or MathL)
28
29

30 - 255 Programmed HALT

(File: TOP2Errors.Text)
