

*Rm 217 Brouse Copy*

NUMBER 22 & 23

Pascal Users Group

# Pascal News

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCALERS

SEPTEMBER, 1981

Two for one ...



Or one for two?

Return to:

Pascal Users Group  
P.O. Box 4406  
Allentown, Pa. 18104-4406

Return postage guaranteed  
Address Correction requested



ATTN: ROOM 217 BROUSE COPY [81]  
UNIV. OF MINNESOTA  
UCC : 227EX

MAR 24 1982

POLICY: PASCAL NEWS

(15-Sep-80)

- \* Pascal News is the official but informal publication of the User's Group.
- \* Pascal News contains all we (the editors) know about Pascal; we use it as the vehicle to answer all inquiries because our physical energy and resources for answering individual requests are finite. As PUG grows, we unfortunately succumb to the reality of:

1. Having to insist that people who need to know "about Pascal" join PUG and read Pascal News - that is why we spend time to produce it!

2. Refusing to return phone calls or answer letters full of questions - we will pass the questions on to the readership of Pascal News. Please understand what the collective effect of individual inquiries has at the "concentrators" (our phones and mailboxes). We are trying honestly to say: "We cannot promise more that we can do."

\* Pascal News is produced 3 or 4 times during a year; usually in March, June, September, and December.

\* ALL THE NEWS THAT'S FIT, WE PRINT. Please send material (brevity is a virtue) for Pascal News single-spaced and camera-ready (use dark ribbon and 18.5 cm lines!)

\* Remember: ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.

\* Pascal News is divided into flexible sections:

POLICY - explains the way we do things (ALL-PURPOSE COUPON, etc.)

EDITOR'S CONTRIBUTION - passes along the opinion and point of view of the editor together with changes in the mechanics of PUG operation, etc.

HERE AND THERE WITH PASCAL - presents news from people, conference announcements and reports, new books and articles (including reviews), notices of Pascal in the news, history, membership rosters, etc.

APPLICATIONS - presents and documents source programs written in Pascal for various algorithms, and software tools for a Pascal environment; news of significant applications programs. Also critiques regarding program/algorithm certification, performance, standards conformance, style, output convenience, and general design.

ARTICLES - contains formal, submitted contributions (such as Pascal philosophy, use of Pascal as a teaching tool, use of Pascal at different computer installations, how to promote Pascal, etc.).

OPEN FORUM FOR MEMBERS - contains short, informal correspondence among members which is of interest to the readership of Pascal News.

IMPLEMENTATION NOTES - reports news of Pascal implementations: contacts for maintainers, implementors, distributors, and documentors of various implementations as well as where to send bug reports. Qualitative and quantitative descriptions and comparisons of various implementations are publicized. Sections contain information about Portable Pascals, Pascal Variants, Feature-Implementation Notes, and Machine-Dependent Implementations.

----- ALL-PURPOSE COUPON ----- (15-Dec-81)

Pascal Users Group  
P.O. Box 4406  
Allentown, Pa. 18104-4406 USA

**\*\*Note\*\***

- We will not accept purchase orders.
- Make checks payable to: "Pascal Users Group", drawn on a U.S. bank in U.S. dollars.
- Note the discounts below, for multi-year subscription and renewal.
- The U. S. Postal Service does not forward Pascal News.

- |  |             | USA   | UK   | Europe | Aust.  |
|--|-------------|-------|------|--------|--------|
| [ ] Enter me as a new member for:  | [ ] 1 year  | \$10. | #6.  | DM20.  | A\$8.  |
| [ ] Renew my subscription for:   | [ ] 2 years | \$18. | #10. | DM45.  | A\$15. |
|  | [ ] 3 years | \$25. | #15. | DM50.  | A\$20. |
| [ ] Send Back Issue(s)   | _____       |       |      |        |        |
| [ ] My new address/phone is listed below   |             |       |      |        |        |
| [ ] Enclosed please find a contribution, idea, article or opinion which is submitted for publication in the <u>Pascal News</u> . |             |       |      |        |        |
| [ ] Comments:  | _____       |       |      |        |        |

ENCLOSED PLEASE FIND:
CHECK no. _____

NAME \_\_\_\_\_

ADDRESS \_\_\_\_\_

PHONE \_\_\_\_\_

COMPUTER \_\_\_\_\_

DATE \_\_\_\_\_

## JOINING PASCAL USERS GROUP?

Membership is open to anyone: Particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan. Please enclose the proper prepayment (check payable to "Pascal User's Group"); we will not bill you. Please do not send us purchase orders; we cannot endure the paper work! When you join PUG any time within a year: January 1 to December 31, you will receive all issues of Pascal News for that year. We produce Pascal News as a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News. We desire to minimize paperwork, because we have other work to do.

-----

American Region (North and South America) Join through PUG(USA).

European Region (Europe, North Africa, Western Asia): Join through PUG(EUR) Pascal Users Group, c/o Grado Computer Systems & Software, Weissenburgerstrasse 25, D-8000, Munchen 80, Germany.

United Kingdom Region: join through PUG(UK) : Pascal Users Group, c/o Shetlandtel, Walls, Shetland, ZE2 9PF, United Kingdom.

Australasian Region (Australia, East Asia - incl. India & Japan): PUG(AUS). Pascal Users Group, c/o Arthur Sale, Department of Information Science, University of Tasmania, Box 252C GPO, Hobart, Tasmania 7001, Australia. International telephone: 61-02-202374

-----

## RENEWING?

Please renew early (before November) and please write us a line or two to tell us what you are doing with Pascal, and tell us what you think of PUG and Pascal News. Renewing for more than one year saves us time.

## ORDERING BACK ISSUES OR EXTRA ISSUES?

Our unusual policy of automatically sending all issues of Pascal News to anyone who joins within a year means that we eliminate many requests for backissues ahead of time, and we don't have to reprint important information in every issue--especially about Pascal implementations!

Issues 1 .. 8 (January, 1974 - May 1977) are out of print.

Issues 9 .. 12, 13 .. 16, & 17 .. 20 are available from PUG(USA) all for \$15.00 a set, and from PUG(AUS) all for \$A15.00 a set.

Extra single copies of new issues (current academic year) are: \$5.00 each - PUG(USA); and \$A5.00 each - PUG(AUS).

## SENDING MATERIAL FOR PUBLICATION?

Your experiences with Pascal (teaching and otherwise), ideas, letters, opinions, notices, news, articles, conference announcements, reports, implementation information, applications, etc. are welcome. Please send material single-spaced and in camera-ready (use a dark ribbon and lines 18.5 cm. wide) form.

All letters will be printed unless they contain a request to the contrary.

	Pascal News #22 & 23	September 1981	Index
0	POLICY, COUPONS, INDEX, ETC.		
1	EDITORS CONTRIBUTION		
3	HERE AND THERE WITH Pascal		
3	Summary of Implementations (for PN 15..18)		(G. Marshall)
4	APPLICATIONS		
4	The FMI Compiler (code)		A. Tanenbaum
38	Options -- Control Statement Option Settings		S. Leonard
39	Treeprint -- Prints Trees on a Character Printer		Freed & Carosso
44	Compress & Recall -- Text compression using Huffman codes		T. Stone
50	ARTICLES		
50	"The Performance of three CP/M based Translators"		Johnson & Sidebottom
54	"A Geographer Teaches Pascal -- Reflections on the Experience"		J. Pitzl
56	"An Extension That Solves Four Problems"		J. Yavner
61	OPEN FORUM FOR MEMBERS		
68	IMPLEMENTATION NOTES		
81	ONE PURPOSE COUPON, POLICY		

-----

APPLICATION FOR LICENSE TO USE VALIDATION SUITE FOR PASCAL

Name and address of requestor: \_\_\_\_\_  
(Company name if requestor is a company): \_\_\_\_\_  
Phone Number: \_\_\_\_\_  
Name and address to which information should be addressed (write "as above" if the same) \_\_\_\_\_  
Signature of requestor: \_\_\_\_\_  
Date: \_\_\_\_\_

In making this application, which should be signed by a responsible person in the case of a company, the requestor agrees that:

- a) The Validation Suite is recognized as being the copyrighted, proprietary property of R. A. Freak and A. H. J. Sale, and
- b) The requestor will not distribute or otherwise make available machine-readable copies of the Validation Suite, modified or unmodified, to any third party without written permission of the copyright holders.

In return, the copyright holders grant full permission to use the programs and documentation contained in the Validation Suite for the purpose of compiler validation, acceptance tests, benchmarking, preparation of comparative reports and similar purposes, and to make available the listings of the results of compilation and execution of the programs to third parties in the course of the above activities. In such documents, reference shall be made to the original copyright notice and its source.

Distribution Charge: \$50.00

Make checks payable to ANPA/RI in US dollars drawn on a US bank.  
Remittance must accompany application.

Source Code Delivery Medium Specification:

- 800 bpi, 9-track, NRZI, odd parity, 600' magnetic tape  
 1600 bpi, 9-track, PE, odd parity, 600' magnetic tape

ANSI-STANDARD

a) Select Character Code Set:

- ASCII       EBCDIC

b) Each logical record is an 80 character card image. Select block size in logical records per block.

- 40       20       10

Special DEC System Alternates:

- RSX-IAS PIP Format (requires ANSI MAGtape RSX SYSGEN)  
 DOS-RSTS FLX Format

Mail Request to:  
ANPA/RI  
P.O. Box 598  
Easton, Pa. 18042  
USA  
Attn: R. J. Cichelli

Office Use Only

Signed \_\_\_\_\_  
Date \_\_\_\_\_

Richard J. Cichelli  
On behalf of A.H.J. Sale and R.A.Freak

## Editor's Contribution

### GOOFED AGAIN

Yes as all you loyal Pennsylvanians have noticed in the last issue of PN we managed to mess up the zip code of Allentown PA, and of course the USPS has come down on us like a ton of bricks! Please note that the zip is 18014 not 18170. It has been corrected in the new APC.

### THE NEW APC

Speaking of the new APC we have simplified it some more, and added current prices for the UK and Europe, and have modified the reverse side of the coupon to reflect the new foreign editors and their current addresses.

### THE LATEST EUROPEAN SOLUTION

Speaking of the European editors, we have two new ones! One for the UK, and one for the Continent. Nick Hushes will be handling all business for Britain, and Hellmut Heher will be in charge of the European Region. Please see the APC for their addresses.

### ON CALLING

Please restrict yourself to written correspondence when dealing with PUG. This is strictly a scholarly function. None of the editors (including myself) gets paid. All have a real job that pays their bills, and they owe their office hours to their employer. All PUG work is done on their own time. So please write to the appropriate regional editor. It leaves a documentary trail that can be followed and handled as fast as we can. Honest!

### COMBINED ISSUE

This is of course a combined issue. We are doing this to catch up and to beat the postal system and their high rates. If this upsets anyone we are sorry. We are doing our best.

### ON BEING THE EDITOR

Anyone who is interested in being the new editor of PN should write to me at the main address (APC).

### STANDARDS

Good news from the standard front! 7185.1 was approved by the international committee. More next issue from Jim Miner the Standards Editor.

# Here and There With Pascal

## Summary of Implementations

### THIS ISSUE

The highlight of this issue is the long awaited (from last issue at least!) of Andrew Tanenbaum's EM1 compiler. I think it is really great. Tell us what you think! In the Here and There section Gress Marshall has summarized the past few issues (15 .. 19) implementation notes. Thanx. In addition to the EM1 compiler, the Applications section includes an improved version of the subroutine "options", as well as a tree printing routine, and a set of routines to compress and expand text using Huffman codes. Good work! And finally the articles section has some fine contributions. Many people have asked (on the phone ... see above) about how the various CP/M compilers stack up. Now we have an answer. Also there is an article of the experiences of a novice teaching Pascal. From a geography teacher no less! And finally a probins article by Jonathan Yaxner concerning problems with Pascal and some proposals for their solution.

Hope you like it.

Rick

ALL	#15:101	Pascal I (Derived from Pascal S)	
BESM-6	#15:107		
Burroughs B5700	#15:107		
Burroughs B6700/B7700 (MCP)	#19:113		
CDC 6000	#19:115		
CDC 6000	#15:108		
Cyber 70 and 170	#15:108		
DEC PDP-11	#19:115	UCSD Pascal	
DEC PDP-11	#15:111		
DEC PDP-11	#15:112	UCSD Pascal	
DEC PDP-11	#15:124		
DEC PDP-11 (RSTS)	#15:100	Pascal S	
DEC PDP-11 (RSX-11M/IAS)	#17:86		
DEC PDP-11 (RSX-11M/RT-11)	#15:101	Concurrent Pascal	
DEC PDP-11 (Unix)	#15:111		
DEC PDP-11 (Unix)	#15:100	Pascal E	
DEC PDP-11 (Unix)	#15:103	Modula	
DEC PDP-15	#15:124		
DEC VAX	#17:89		
DEC VAX (Unix)	#19:115		
DG Eclipse	#17:106		
DG Eclipse (AOS)	#15:110	RDOS, DOS)	
DG Eclipse (AOS)	#15:109		
DG Eclipse (RDOS)	#15:108		
DG Nova (AOS)	#15:110	RDOS, DOS)	
Digico Micro 16E	#15:113		
Facom 230-45S	#15:112		
General Electric GEC4082	#15:113		
Golem B (GOBOS)	#17:104		
HP 1000	#19:116		
Honeywell 6000 (GCOS III)	#15:113		
Honeywell Level 6	#15:113		
IBM 3033	#19:120		
IBM 360/370	#15:114		
IBM 360/370	#15:115		
IBM 370	#17:104		
IBM 370	#19:117		
IBM 370	#15:124		
IBM 370	#17:102		
IBM 370/303x/43xx	#19:117		
IBM Series 1	#19:116		
IBM Series 1	#15:114		
ICL 1900	#15:116		
Intel 8080/8085	#15:119		
Intel 8080/8085	#15:118		
Intel 8080/8085	#15:119		
Intel 8080/8085	#17:102		
Intel 8080/8085	#15:117		
Intel 8080/8085 (CP/M)	#17:105		
Intel 8080/8085 (TRS-80)	#15:100		
Intel 8080/8085 (Northstar)	#15:100		
Intel 8086	#15:119		
Intel 8086	#15:103		
MOS Tech 6502 (Apple)	#15:107		
Modcomp II and IV	#15:120		
		Motorola 6800	#15:120
		Motorola 6800	#19:120
		Motorola 6800	#19:121
		Motorola 6800	#17:102
		Motorola 6800 (Flex)	#15:123
		Motorola 68000	#19:121
		Motorola 6809	#15:103
		Motorola 6809 (MDOS09)	#17:102
		Nord 10 and 100 (Sintran III)	#15:121
		Perkin-Elmer 3220	#15:122
		Perkin-Elmer 7/16	#15:121
		RCA 1802	#17:103
		RCA 1802	#15:122
		Siemens 7.748	#15:124
		Sperry-Univac V77	#15:124
		Texas Instruments 990	#17:101
		Texas Instruments 9900	#15:124
		Zilog Z-80	#15:124
		Zilog Z-80	#19:123
		Zilog Z-80	#15:124
		Zilog Z-80	#17:88
		Zilog Z-80	#17:104
		Zilog Z-80 (CP/M)	#17:103
		Zilog Z-80 (TRS-80)	#15:124
		Zilog Z-80 (TRS-80)	#19:124
		Zilog Z80	#15:118
		Zilog Z80	#15:119
		Zilog Z8000	#15:119

# Applications

## EM1 COMPILER

```

1 #include ".../local.h"
2 #include ".../em1.h"
3
4 (c) copyright 1980 by the Vrije Universiteit, Amsterdam, The Netherlands.
5 Explicit permission is hereby granted to universities to use
6 or duplicate this program for educational or research purposes. All
7 other use or duplication by universities, and all use or duplication
8 by other organizations is expressly prohibited unless written
9 permission has been obtained from the Vrije Universiteit. Requests
10 for such permissions may be sent to
11
12 Dr. Andrew S. Tanenbaum
13 Wiskundig Seminarium
14 Vrije Universiteit
15 Postbox 7161
16 1007 MC Amsterdam
17 The Netherlands
18
19 Organizations wishing to modify part of this software for subsequent
20 sale must explicitly apply for permission. The exact arrangements
21 will be worked out on a case by case basis, but at a minimum
22 will require the organization to include the following notice in all
23 software and documentation based on our work:
24
25 This product is based on the Pascal system
26 developed by Andrew S. Tanenbaum, Johan W. Stevenson
27 and Hans van Steyvers of the Vrije Universiteit, Amsterdam,
28 The Netherlands.
29
30
31 (if next line is included the compiler is written in standard pascal)
32 #define STANDARD 1)
33
34 (if next line is included, then code is produced for segmented memory)
35 #define SEGMENTS 1)
36
37 (author: Johan Stevenson Version: 31)
38 (s: no source line numbers)
39 (s: no subrange checking)
40 (s: no assertion checking)
41 #ifdef STANDARD
42 (s: test conformance to standard)
43 #endif
44
45 program pascal(input,em1,errors);
46
47 ( This Pascal compiler produces EM1 code as described in
48 - A.S.Tanenbaum, J.W.Stevenson & H. van Steyvers,
49 "Description of an experimental machine architecture for use of
50 block structured languages" Informatica rapport 54,
51 - K. Jensen & E. Wirth, PASCAL user manual and report, Springer-Verlag.
52 Several options may be given in the normal pascal way. Moreover,
53 a positive number may be used instead of + and -. The options are:
54 s: interpret assertions (+)
55 o: C-type strings allowed (-)
56 d: type long may be used (-)

```

```

57 f: size of reals in words (2)
58 i: controls the number of bits in integer sets (16)
59 l: insert code to keep track of source lines (+)
60 o: optimize (+)
61 p: size of pointers in words (1)
62 r: check subranges (+)
63 s: accept only standard pascal programs (-)
64 t: trace procedure entry and exit (-)
65 u: treat '-' as letter (-)
66
67 }
68 #ifdef STANDARD
69 label 9999;
70 #endif
71
72 const
73
74 (powers of two)
75 t1 = 128;
76 t2 = 255;
77 t3 = 256;
78 t4 = 16384;
79 t15 = 32767;
80
81 (EM-1 sizes)
82 bytebits = 8;
83 wordbits = 16;
84 wml = 15; (wordbits-1)
85 minint = -t15ml;
86 maxint = t15ml;
87 maxintstring = '0000032767';
88 maxlongstring = '2147483647';
89
90
91 bytesize = 1;
92 wordsize = 2;
93 addrsz = wordsize;
94 punitsize = wordsize;
95 shortsize = wordsize;
96 longsize = 4;
97
98 #ifdef SFLDPT
99 floatsize = 4;
100 #endif
101 #ifdef SFLOAT
102 floatsize = 8;
103 #endif
104
105 (Pascal sizes, for pchar, real, int and float see handleopt)
106 ( EM-1 requires that objects greater than a single byte start at a
107 word boundary, so their address is even. Normally, a full word
108 is also allocated for objects of a single byte. This extra byte
109 is really allocated to the object, not only stripped by alignment,
110 i.e. if the value false is assigned to a boolean variable then
111 both bytes are cleared. For single byte objects in packed arrays
112 or packed records, however, only one byte is allocated, even if
113 the next byte is unused. Strings are packed arrays. The size of
114 pointers is 2 by default, but can be changed at runtime by the

```

PASCAL NEWS #22 & #23

SEPTEMBER, 1981

Page 4

PASCAL NEWS #22 & #23

SEPTEMBER, 1981

Page 5

```

113 p-option. Floating point numbers in EM-1 currently have size 4,
114 but this might change in the future to 8. The default can be
115 overridden by the f-option. The routines involved with alignment
116 are 'even', 'address' and 'arraysize'.
117
118 )
119
120 boolsize = bytesize;
121 charsize = bytesize;
122 intsize = shortsize;
123 buffsize = 512;
124 maxsetsize = 4096; (t15 div bytebits)
125
126 (maximal indices)
127 lmax = 8;
128 fmax = 14;
129 smax = 72;
130 rmax = 72;
131 lmax = 10;
132
133 (opt values)
134 off = 0;
135 on = 1;
136
137 (for push and pop)
138 global = false;
139 local = true;
140
141 (set bounds)
142 misetbit = 0;
143 misetint = 15; (default)
144
145 (constants describing the compact EM1 code)
146 MAGICLOW = 172;
147 MAGICHIGH = 0;
148 noserror = 0;
149 nosopt = 1;
150 nosoptimal = 2;
151 nosreg = 3;
152 nosline = 4;
153 nosfloats = 5;
154
155 (ASCII characters)
156 Lab = 9;
157 newline = 10;
158 crlf = 13;
159 eof = 12;
160 error = 13;
161
162 (miscellaneous)
163 maxseg = 127; (maximal segment number)
164 maxobord = 127; (maximal ordinal number of obars)
165 maxarg = 13; (maximal index in argv)
166 radix = 34; (number of reserved words)
167 space = ' ';
168 emptyform = ' '

```

```

169 type
170 (scalar types)
171 symbol = (comma,semicolon,colon,colon2,notat,brack,ident,
172 intoct,charoct,realoct,longoct,stringoct,alloc,minint,
173 plusy,parent,arrow,array,record,sets,files,
174 packeday,progr,labels,constat,types,var,proc,
175 funcy,beginsy,gotosy,ifsy,whilesy,repeaty,for,
176 withy,casey,becomes,staray,divey,mody,aliasy,
177 array,aray,aray,aray,aray,aray,aray,
178 leaf,insy,andsy,eisay,untilsy,ofsy,dosy,
179 downtosy,cosy,thesy,rbrack,rparent,period
180 ); (the order is important)
181 charctype = (lower,upper,digit,laport,tabch,
182 quotch,dquotch,odquotch,periodch,leasch,
183 greaterch,lparentch,lbracch,
184 ); (different entries)
185 rparentch,lbrackch,rbrackch,ocommach,semich,arrowch,
186 plusch,minch,alsh,star,equal,
187 ); (also symbols)
188 others
189 );
190
191 standpf = (read,readln,write,writeln,pput,pgot,
192 pstart,proccite,pgnes,pdispose,ppack,punpack,
193 pmack,prelease,ppage,phalt,
194 ); (all procedures)
195 feof,foeln,fabs,fsg,ford,fobr,fpre,fuoc,fodd,
196 frumo,fround,fsin,fcos,fexp,fsgt,fln,farctan
197 ); (all functions)
198 ); (the order is important)
199
200 libnames = (IN, OUT, CLS, WML, (input and output)
201 OF, GET, END, DDC, EME, NML, NLE (see input files)
202 CDE, FPU, WEI, MBI, WAC, WSC, WRS, WSS, WML,
203 WMB, WMR, WMR, WML, WSL, WMP, WR1, WSL, WML, PAC,
204 ); (on output files, order important)
205 ABS, END, SIN, COS, EXP, SQRT, LOG, ATN
206 ); (floating point)
207 ABI, ABL, BCP, BVS, HWV, SAV, EST, IBI, IBL,
208 ASS, OTO, PAC, WMP, DIS, ARZ, HDI, HDL
209 ); (miscellaneous)
210
211
212 structform = (scalar,subrange,pointer,power,files,array,array,
213 records,variant,tag); (order important)
214
215 structflag = (speak,withfile);
216
217 identflag = (refer,used,assigned,over,assigned);
218
219 declsize = (types,lower,vers,field,over,proc,func);
220
221 kindofp = (standard,formal,actual,extra,forward);
222
223 where = (blk,rec,wrow);
224
225 strkind = (out,fixed,pfixed,loaded,ploded,indented);
226
227 tuostruc = (eq,subeq,lr,rl,il,il,lr,lr,le,le,se,se,noted);
228
229
230 (subrange types)
231 sgrens = 0..maxseg;
232 idrens = 1..lmax;
233 furangs = 1..fmax;

```

225 rrange= 0..rval; 281 end;

226 bytes 0..lbit;

228 (pointer types)

229 sp= "structure; 284

230 ip= "identifier; 285 fname:ip; (one deeper)

231 l= "labl; 286 (first name: root of tree)

232 bps= "blockinfo; 287 case occur:where of

233 sp= "nameinfo; 288 blok:();

289 rec: ();

290 arec:(w:sttr) (name space opened by with statement)

291 end;

292 blockinfo:record (all info of the current procedure)

293 size:ip; (pointer to blockinfo of surrounding proc)

294 lc:integer; (data location counter (from begin of proc))

295 lbno:integer; (number of last local label)

296 forwcount:integer; (number of not yet specified forward procs)

297 lchain:ip; (first label: header of chain)

298 end;

300 structure:record

301 size:integer; (size of structure in bytes)

302 sflag:flagset; (flag bits)

303 case form:structform of

304 scalar : (scaino:integer; (number of range descriptor)

305 fconst:ip; (names of constants)

306 );

307 subrange:(ain,am:integer; (lower and upper bound)

308 ranetype:ip; (type of bounds)

309 subno:integer; (number of subr descriptor)

310 );

311 pointer : (sttype:sp; (type of pointed object)

312 power : (dast:sp; (type of set elements)

313 files : (f:filetype:sp; (type of file elements)

314 arrays,orarray: (type of array elements)

315 inttype:sp; (type of array index)

316 arpos:position; (position of array descriptor)

317 );

318 records : (fstfid:ip; (points to first field)

319 tagap:sp; (points to tag if present)

320 );

321 variant : (varval:integer; (tag value for this variant)

322 mtrvar:sp; (next equilevel variant)

323 subtag:sp; (points to tag for sub-case)

324 );

325 tag : (fstvar:sp; (first variant of case)

326 tfid:sp; (type of tag)

327 );

328 end;

329 end;

331 identifier:record

332 idtype:sp; (type of identifier)

333 name:alpha; (name of identifier)

334 link,plink:ip; (see enterid,searchid)

335 next:ip; (used to make several chains)

336 iflag:flagset; (several flag bits)

337 case klass:ideclass of

338 types : ();

339 konst : (value:integer); (for integers the value is

340 computed and stored in this field.

341 For strings and reals an assembler constant is

342 defined labeled '1', '2', ...

343 This '1' number is then stored in value.

344 For reals value may be negated to indicate that

345 the opposite of the assembler constant is needed.)

346 vars : (vpos:position); (position of var)

347 field : (ffoffset:integer); (offset to begin of record)

348 oarrbnd : (); (idtype points to array)

349 proc,func

350 (name pf:kindofpf of

351 standard:(key:stastandpf); (identification)

352 formal,actual,forward,extra: (lv gives declaration level.

353 pfpos:position; (lv gives instruction segment of this proc and

354 is relevant for formal pf's and for

355 functions (no conflict)).

356 for functions: ad is the result address.

357 for formal pf's: ad is the address of the

358 descriptor)

359 pfno:integer; (unique pf number)

360 parhead:ip; (head of parameter list)

361 head:integer (1s when heading summed)

362 );

363 end;

364

365

367 labl:record

368 next:ip; (chain of labels)

369 seen:boolean;

370 labval:integer; (label number given by the programmer)

371 labname:integer; (label number given by the compiler)

372 labid:integer (same name only locally used,

373 otherwise dibno of label information)

374 end;

375

376

377 var (the most frequent used externals are declared first)

378 sy:symbol; (last symbol)

379 sstr; (type,access method,position,value of expr)

380 (returned by inqpr)

381 ch:char; (last character)

382 chtype:chtype; (type of ch, used by inqpr)

383 val:integer; (if last symbol is an constant)

384 (string length)

385 col:boolean; (true if current ch replaces a newline)

386 newstrng:boolean; (true for strings in " ")

387 id:alpha; (if last symbol is an identifier)

388 (same counters)

389 line:integer; (line number on code file (1..n))

390 dibno:integer; (number of last global number)

391 lmax:integer; (deepest level of nesting of lv)

392 level:integer; (current static level)

393 ptrsize:integer;

394 realize:integer; (file header size)

395 rsize:integer; (index in arg)

396 lastpfno:integer; (unique pf number counter)

397 oopt:integer; (C-type strings allowed if on)

398 dopt:integer; (longs allowed if on)

399 lopt:integer; (number of bits in sets with base integer)

400 sop:integer; (standard option)

401 (pointers pointing to standard types)

402 realptr,inp:tr,txtptr,emptyst,boolptr:sp;

403 charptr,nilptr,stringptr,longptr:sp;

404

405 (flags)

406 give:line:boolean; (give source line number at next statement)

407 including:boolean; (no LHM's for included code)

408 eof:expected:boolean; (quit without error if true (nextch))

409 main:boolean; (complete programme or a module)

410 inttypedec:boolean; (true if nested in typedefinition)

411 flused:boolean; (true if floating point instructions are used)

412 seconddot:boolean; (indicates the second dot of '...')

413 (pointers)

414 fptr:ip; (head of chain of forward reference pointers)

415 progid:ip; (program identifier)

416 currproc:ip; (current proc/func ip (see casestatement))

417 top:ip; (pointer to the most recent name space)

418 lastsp:ip; (pointer to nameinfo of last searched ident)

419 (records)

420 b:blockinfo; (all info to be checked at pfdeclaration)

421 e:errrec; (all info required for error messages)

422 f:fsttr; (fsttr for current file name)

423 (arrays)

424 source:fttype; (name of pascal source file)

425 strbuf:array[1..max] of char;

426 lop:array[boolean] of ip;

427 (pfno:standard input, true:standard output)

428 rv:array(r:range) of alpha; (reserved words)

429 (reserved words)

430 frv:array(0..idmax) of integer;

431 (indices in rv)

432 (symbols)

433 rsy:array(r:range) of symbol;

434 (symbol for reserved words)

435 (characters)

436 ch:array(char) of chartype;

437 (ch:char of a character)

438 (symbols)

439 chsy:array(r:parmatch, equal) of symbol;

440 (symbol for single character symbols)

441 lms:array(l:ldmax) of ip;

442 (mnemonics of pascal library routines)

443 opt:array('a'..'z') of integer;

444 (different options)

445 foroopt:array('a'..'z') of boolean;

446 (used in searchid)

447 wdefip:array(ideclass) of ip;

448 (used in searchid)

449 (records)

450 rrv:array(0..maxrarg) of

451 record name:alpha; ad:integer end;

452 (here here the external heading names)

453 (files)

```
449  eml:file of byte;  (the EM1 code)
450  errors:file of error;
451  (the compilation errors)
452  (=====)
454  procedure gen2bytes(b:byte; i:integer);
455  var b1,b2:byte;
456  begin
457  if i<0 then
458  if i<minint then begin b1:=0; b2:=7 end
459  else begin i:=i-1; b1:=tda1 - i mod td; b2:=tda1 - i div td end
460  else begin b1:=i mod td; b2:=i div td end;
461  write(eml,b1,b2);
462  end;
464  procedure genct(i:integer);
465  begin
466  if (i>0) and (i<ap_max0) then write(eml,i,sp_fct0)
467  else gen2bytes(sp_ct2,i)
468  end;
470  procedure genclb(i:integer);
471  begin if i<t8 then write(eml,sp_1lb1,i) else gen2bytes(sp_1lb2,i) end;
473  procedure genilb(i:integer);
474  begin lino:=lino+1;
475  if i<ap_nilb0 then write(eml,i,sp_filb0) else genclb(i);
476  end;
478  procedure genlb(i:integer);
479  begin if i<t8 then write(eml,sp_dlb1,i) else gen2bytes(sp_dlb2,i) end;
481  procedure gen0(b:byte);
482  begin write(eml,b); lino:=lino+1 end;
484  procedure gen1(b:byte; i:integer);
485  begin gen0(b); genct(i) end;
487  procedure gen2(b:byte; d:integer);
488  begin gen0(b); genlb(d) end;
490  procedure genident(name:type; var a:alpha);
491  var i,j:integer;
492  begin i:=ldm;
493  while (a[i]=' ') and (i>1) do i:=i-1;
494  write(eml,name,i);
495  for j:=1 to i do write(eml,ord(a[j]));
496  end;
498  procedure genap(a:libname);
499  var i:integer;
500  begin gen0(sp_cal); write(eml,sp_pnm,a);
501  for i:=1 to 4 do write(eml,ord(lanm[i]));
502  end;
504  procedure genpnm(b:byte; fil:ip);
```

```
505  var n:alpha; i,j:integer;
506  begin
507  if fil<sp_pos.lv<1 then n:=fil.p_name else
508  n:=fil.p_name; j:=1; i:=fil.p_fno;
509  while i<0 do
510  begin j:=j+1; n[j]:=chr(1 mod 10 + ord('0')); i:=i div 10 end;
511  end;
512  gen0(b); genident(sp_pnm,n)
513  end;
515  procedure genend;
516  begin write(eml,sp_cnd) end;
518  procedure genlin;
519  begin give_lino:=false;
520  if opt['!']<off then if main then gen!(op_lino,orig)
521  end;
523  procedure genreg(ad,az,nr:integer);
524  begin
525  if az<wordsize then
526  begin gen!(pa_mes,mesreg); genct(ad); genct(nr); genend end
527  end;
529  (=====)
531  procedure puterr(err:integer);
532  (as you will notice, all error numbers are preceded by 'e' and '0' to
533  ease their renumbering in case of new error numbers.
534  )
535  begin e:=err; write(errors,e);
536  if err>0 then begin gen!(pa_mes,meserror); genend end
537  end;
539  procedure error(err:integer);
540  begin e:=sp_spaces; e.mes:= -1; puterr(err) end;
542  procedure errid(err:integer; var id:alpha);
543  begin e:=sp_id; e.mes:= -1; puterr(err) end;
545  procedure errint(err:integer; i:integer);
546  begin e.mes:=i; e.mes:=sp_spaces; puterr(err) end;
548  procedure aspperr(err:integer);
549  begin if a.sp<off then begin error(err); a.sp:=nil end end;
551  procedure teststand;
552  begin if opt<off then error(-e01) end;
554  procedure enterid(fil: ip);
555  (enter id pointed at by fil into the name-table,
556  which on each declaration level is organized as
557  an unbalanced binary tree)
558  var n:alpha; lip,rip:ip; lleft,again:boolean;
559  begin n:=fil.p_name; again:=false;
560  lip:=top.p_fname;
```

```
561  if lip=nil then top.p_fname:=fil else
562  begin
563  repeat lip:=lip;
564  if lip.p_name<n then
565  begin lip:=lip.llink; lleft:=true end
566  else
567  begin if lip.p_name=n then again:=true; (name conflict)
568  lip:=lip.rlink; lleft:=false;
569  end;
570  until lip=nil;
571  if lleft then lip:=lip.llink:=fil else lip:=lip.rlink:=fil
572  end;
573  fil.llink:=nil; fil.rlink:=nil;
574  if again then errid(-e02,n);
575  end;
577  procedure initpos(var p:position);
578  begin p.lv:=level; p.ad:=0;
579  if def SEGMENTS
580  p.ad:=0
581  #endif
582  end;
584  procedure initf(fp:tp; fd:integer);
585  begin with a do begin
586  asp:=fp; packbit:=false; ak:=fixed; pos.ad:=fd; pos.lv:=level;
587  if def SEGMENTS
588  pos.ag:=0;
589  #endif
590  end end;
592  function newp(kl:ideclass; n:alpha; id:tp; actip:ip);
593  var p:ip; fl:flagset;
594  begin fl:=[];
595  case kl of
596  types,ovrbed: (similar structure)
597  new(p,types);
598  konst:
599  begin new(p,konst); p.value:=0 end;
600  vars:
601  begin new(p,vars); p:=used,assigned; initpos(p,vpos) end;
602  field:
603  begin newp(field); p.p_offset:=0 end;
604  proc_func: (same structure)
605  begin newp(proc,actual); p.p_kind:=actual;
606  initpos(p,p_pos); p.p_fno:=0; p.p_parhead:=nil; p.p_head:=0
607  end;
608  end;
609  p.p_name:=n; p.p_klass:=kl; p.p_idtype:=id; p.p_next:=ent;
610  p.llink:=nil; p.rlink:=nil; p.lflag:=f; newp:=p
611  end;
613  function newp(sf:structform; sz:integer);
614  var p:ip; sf:flagset;
615  begin sf:=[];
616  case sf of
```

```
617  scalar:
618  begin new(p,scalar); p.p_scalar:=0; p.p_fconst:=nil end;
619  subrange:
620  new(p,subrange);
621  pointer:
622  begin new(p,pointer); p.p_ctype:=nil end;
623  power:
624  new(p,power);
625  files:
626  begin newp(files); sf:=[]; with file end;
627  arrays,ovrarr: (same structure)
628  new(p,arrays);
629  records:
630  new(p,records);
631  variant:
632  new(p,variant);
633  tag:
634  new(p,tag);
635  end;
636  p.p_fors:=f; p.p_size:=sz; p.p_sflag:=sf; newp:=p;
637  end;
639  procedure initf;
640  var o:char;
641  begin
642  (initialize the first name space)
643  new(top,blk); top.p_occu:=ak; top.llink:=nil; top.p_fname:=nil;
644  level:=0;
645  (reserved words)
646  rw[0]:='if'; rw[1]:='do'; rw[2]:='of';
647  rw[3]:='to'; rw[4]:='in'; rw[5]:='on';
648  rw[6]:='end'; rw[7]:='for'; rw[8]:='nil';
649  rw[9]:='var'; rw[10]:='div'; rw[11]:='mod';
650  rw[12]:='set'; rw[13]:='and'; rw[14]:='not';
651  rw[15]:='then'; rw[16]:='else'; rw[17]:='with';
652  rw[18]:='case'; rw[19]:='type'; rw[20]:='goto';
653  rw[21]:='file'; rw[22]:='begin'; rw[23]:='until';
654  rw[24]:='while'; rw[25]:='array'; rw[26]:='const';
655  rw[27]:='label'; rw[28]:='repeat'; rw[29]:='record';
656  rw[30]:='down to'; rw[31]:='packed'; rw[32]:='program';
657  rw[33]:='function'; rw[34]:='procedure';
658  (corresponding symbols)
659  rsf[0]:='if'; rsf[1]:='do'; rsf[2]:='of';
660  rsf[3]:='to'; rsf[4]:='for'; rsf[5]:='or';
661  rsf[6]:='and'; rsf[7]:='for'; rsf[8]:='nil';
662  rsf[9]:='var'; rsf[10]:='div'; rsf[11]:='mod';
663  rsf[12]:='set'; rsf[13]:='and'; rsf[14]:='not';
664  rsf[15]:='then'; rsf[16]:='else'; rsf[17]:='with';
665  rsf[18]:='case'; rsf[19]:='type'; rsf[20]:='goto';
666  rsf[21]:='file'; rsf[22]:='begin'; rsf[23]:='until';
667  rsf[24]:='while'; rsf[25]:='array'; rsf[26]:='const';
668  rsf[27]:='label'; rsf[28]:='repeat'; rsf[29]:='record';
669  rsf[30]:='down to'; rsf[31]:='packed'; rsf[32]:='program';
670  rsf[33]:='function'; rsf[34]:='procedure';
671  (indices into rw to find reserved words fast)
672  frw[0]:=0; frw[1]:=0; frw[2]:=6; frw[3]:=15; frw[4]:=22;
```



```

673   frm(5):=28; frm(6):=32; frm(7):=33; frm(8):=35;
674   (char types)
675   for c:=chr(0) to chr(maxcharord) do os(c):=others;
676   for c:=0 to 9 do os(c):=digit;
677   for c:=a to z do os(c):=upper;
678   for c:=A to Z do os(c):=lower;
679   os(chr(maxline))::=layout;
680   os(chr(maxtab))::=layout;
681   os(chr(maxfeed))::=layout;
682   os(chr(maxret))::=layout;
683   (characters with corresponding char type in ASCII order)
684   os(chr(tab))::=taboh;
685   os(' ')::=space; os('!')::=excl; os('@')::=at; os('#')::=hash;
686   os('$')::=dollar; os('%')::=pcent; os('&')::=amp; os('\'')::=apost;
687   os('(')::=lpar; os(')')::=rpar; os('*')::=ast; os('~')::=tilde;
688   os('`')::=backtick; os('^')::=circ; os('&#x2013;')::=dash;
689   os('_')::=under; os('`')::=grave; os('~')::=asciitilde;
690   os('`')::=grave; os('~')::=asciitilde;
691   os('~')::=asciitilde;
692   (single character symbols in char type order)
693   os('`')::=grave; os('~')::=asciitilde;
694   os('~')::=asciitilde;
695   os('~')::=asciitilde;
696   os('~')::=asciitilde;
697   os('~')::=asciitilde;
698   os('~')::=asciitilde;
699   os('~')::=asciitilde;
700   os('~')::=asciitilde;
701   os('~')::=asciitilde;
702   os('~')::=asciitilde;
703   os('~')::=asciitilde;
704   os('~')::=asciitilde;
705   os('~')::=asciitilde;
706   os('~')::=asciitilde;
707   os('~')::=asciitilde;
708   os('~')::=asciitilde;
709   os('~')::=asciitilde;
710   os('~')::=asciitilde;
711   os('~')::=asciitilde;
712   os('~')::=asciitilde;
713   os('~')::=asciitilde;
714   os('~')::=asciitilde;
715   os('~')::=asciitilde;
716   os('~')::=asciitilde;
717   os('~')::=asciitilde;
718   os('~')::=asciitilde;
719   os('~')::=asciitilde;
720   os('~')::=asciitilde;
721   os('~')::=asciitilde;
722   os('~')::=asciitilde;
723   os('~')::=asciitilde;
724   os('~')::=asciitilde;
725   os('~')::=asciitilde;
726   os('~')::=asciitilde;
727   os('~')::=asciitilde;
728   os('~')::=asciitilde;

```

```

729   charptr:=fontst;sp;
730   end;
731
732   procedure init3;
733   var j:standp; p:ip; q:mp;
734   p:=array(standp) of alpha;
735   ftype:=array(feof..farctan) of sp;
736   begin
737   (names of standard procedures/functions)
738   p[read]:=read; p[readln]:=readln;
739   p[write]:=write; p[writeln]:=writeln;
740   p[put]:=put; p[putln]:=putln;
741   p[page]:=page; p[pgot]:=pgot;
742   p[rewrite]:=rewrite; p[reset]:=reset;
743   p[dispose]:=dispose; p[pack]:=pack;
744   p[unpack]:=unpack; p[mark]:=mark;
745   p[release]:=release; p[halt]:=halt;
746   p[for]:=for; p[forall]:=forall;
747   p[abs]:=abs; p[ord]:=ord;
748   p[and]:=and; p[or]:=or;
749   p[not]:=not; p[xor]:=xor;
750   p[div]:=div; p[mod]:=mod;
751   p[round]:=round; p[trunc]:=trunc;
752   p[frac]:=frac; p[exp]:=exp;
753   p[ln]:=ln; p[log]:=log;
754   p[arctan]:=arctan;
755   (parameter types of standard functions)
756   ftype[feof]:=nil; ftype[faein]:=nil;
757   ftype[faout]:=nil; ftype[faqr]:=nil;
758   ftype[ford]:=nil; ftype[forh]:=nil;
759   ftype[food]:=nil; ftype[foouc]:=nil;
760   ftype[foodf]:=nil; ftype[foouc]:=nil;
761   ftype[foouf]:=nil; ftype[foouc]:=nil;
762   ftype[foouf]:=nil; ftype[foouc]:=nil;
763   ftype[foouf]:=nil; ftype[foouc]:=nil;
764   ftype[foouf]:=nil; ftype[foouc]:=nil;
765   ftype[foouf]:=nil; ftype[foouc]:=nil;
766   (standard procedure/function identifiers)
767   for j:=read to halt do
768   begin new(p.proc.standp); p.klaas:=proc;
769   p.name:=p[feof]; p.p.fkind:=standp; p.key:=j; enterid(p);
770   end;
771   for i:=feof to farctan do
772   begin new(p.func.standp); p.klaas:=func; p.idtype:=ftype[i];
773   if i used not for result type but for parameter type ii
774   p.name:=p[feof]; p.p.fkind:=standp; p.key:=j; enterid(p);
775   end;
776   (program identifier)
777   prog:=newid('main', 'all', nil);
778   (new name space for user external)
779   new(blck); q:=occur:blk; q.allink:=top; q.fname:=nil; top:=q;
780   end;
781
782   procedure init4;
783   var c:char;
784   (pascal library mnemonic)

```

```

785   lan[ELN]:=e'ole'; lan[EFL]:=e'ofl'; lan[CLS]:=c'ole';
786   lan[EM] := e'm';
787   lan[OPW] := o'pog'; lan[ORX] := o'get'; lan[ROX] := r'rdi';
788   lan[RDC] := r'rdc'; lan[RDE] := r'rdr'; lan[RDL] := r'rdl';
789   lan[RIM] := r'rio';
790   lan[CMX] := c'm'; lan[PTX] := p'pct'; lan[WR] := w'wr';
791   lan[WS] := w's'; lan[WC] := w'wrc'; lan[WSC] := w'wsc';
792   lan[WSB] := w'wsb'; lan[WSM] := w'wsm'; lan[WSR] := w'wrr';
793   lan[WSL] := w'wsl'; lan[WSU] := w'wsu'; lan[WSV] := w'wsv';
794   lan[WSW] := w'wsd'; lan[WSX] := w'wsx'; lan[WSY] := w'wsy';
795   lan[WSZ] := w'wsz'; lan[WSA] := w'wsa'; lan[WSB] := w'wsb';
796   lan[WSM] := w'wsd'; lan[WSX] := w'wsx'; lan[WSY] := w'wsy';
797   lan[WSZ] := w'wsz'; lan[WSA] := w'wsa'; lan[WSB] := w'wsb';
798   lan[WSM] := w'wsd'; lan[WSX] := w'wsx'; lan[WSY] := w'wsy';
799   lan[WSZ] := w'wsz'; lan[WSA] := w'wsa'; lan[WSB] := w'wsb';
800   lan[WSM] := w'wsd'; lan[WSX] := w'wsx'; lan[WSY] := w'wsy';
801   lan[WSZ] := w'wsz'; lan[WSA] := w'wsa'; lan[WSB] := w'wsb';
802   lan[WSM] := w'wsd'; lan[WSX] := w'wsx'; lan[WSY] := w'wsy';
803   lan[WSZ] := w'wsz'; lan[WSA] := w'wsa'; lan[WSB] := w'wsb';
804   lan[WSM] := w'wsd'; lan[WSX] := w'wsx'; lan[WSY] := w'wsy';
805   lan[WSZ] := w'wsz'; lan[WSA] := w'wsa'; lan[WSB] := w'wsb';
806   (options)
807   for c:=a to z do begin opt[c]:=0; forceopt[c]:=false end;
808   opt['a']:=true;
809   opt['r']:=floatsize div wordsize; (default real size in words)
810   opt['i']:=maxint div 2;
811   opt['l']:=nil;
812   opt['p']:=true;
813   opt['s']:=addresssize div wordsize; (default pointer size in words)
814   opt['t']:=true;
815   opt:=off;
816   (scalar variables)
817   b.numb:=nil;
818   b.lino:=0;
819   b.lino:=0;
820   b.lino:=0;
821   b.lino:=0;
822   e.lino:=0;
823   e.lino:=0;
824   e.lino:=0;
825   e.lino:=0;
826   e.lino:=0;
827   e.lino:=0;
828   lino:=0;
829   d.lino:=0;
830   arg:=0;
831   lan[pr]:=0;
832   g.lino:=true;
833   lan[ing]:=false;
834   lan[ing]:=false;
835   lan[ing]:=false;
836   lan[ing]:=false;
837   lan[ing]:=false;
838   lan[ing]:=false;
839   lan[ing]:=false;
840   arg[0].ed:=1;

```

```

841   arg[1].ed:=1;
842   end;
843
844   procedure handleopt;
845   begin
846   opt:=opt['a'];
847   opt:=opt['r'];
848   opt:=opt['i'];
849   opt:=opt['l'];
850   opt:=opt['p'];
851   opt:=opt['s'];
852   opt:=opt['t'];
853   opt:=opt['t'];
854   if opt<off then begin opt:=off; opt:=off end;
855   else if opt='u' then opt:=lower;
856   if opt<off then enterid(newid('long', 'long', nil));
857   if opt<off then enterid(newid('string', 'string', nil));
858   if opt<off then begin gen(p_mes, mesoptoff); genend end;
859   if p_mes<wordsize then begin gen(p_mes, mesvirtual); genend end;
860   if opt<off then ftype:=true; (temporary kludge)
861   end;
862
863   (=====)
864
865   procedure trace(name:alpha; fip:ip; var numb:integer);
866   var i:integer;
867   begin
868   if opt['t']<off then
869   begin
870   if numb=0 then
871   begin d.lino:=d.lino+1; numb:=d.lino; genlb(d.lino);
872   gen0(p_mes, numb); write(m_l, sp, numb);
873   for i:=1 to 8 do write(m_l, ord(fip[i].name[i])); genend;
874   end;
875   gen0(sp, numb); gen0(sp, numb);
876   gen0(sp, numb); gen0(sp, numb);
877   end;
878   end;
879
880   function formof(fsp:sp; form:formset):boolean;
881   begin if fsp=nil then formof:=false else formof:=fsp.form in form end;
882   end;
883
884   function sioof(fsp:sp):integer;
885   var s:integer;
886   begin s:=0;
887   if fsp<off then s:=fsp.size;
888   if s<0 then if odd(s) then s:=s+1;
889   end;
890
891   function even(i:integer):integer;
892   begin if odd(i) then i:=i+1; even:=i end;
893   end;
894
895   procedure exchange(i1, i2:integer);
896   var d1, d2:integer;
897   begin d1:=i1-1; d2:=i2-1;

```

```
897   if (d1<0) and (d2<0) then
898     begin gen!(ps_esc,d1); gen!(d2) end
899   end;
901   procedure setop(s:byte);
902   begin gen!(s,even(sizeof(s.asp))) end;
904   procedure spendemptyset(fsp:sp);
905   var i:integer;
906   begin
907     for i:=2 to sizeof(fsp) div wordsize do gen!(op_loc,0); a.asp:=fsp
908   end;
910   procedure push(local:boolean; ad:integer; sz:integer);
911   begin assert not odd(sz);
912     if sz=wordsize then
913       begin if local then gen!(op_lal,ad) else gen!(op_lae,ad);
914         gen!(op_loi,sz)
915       end
916     else
917       if local then gen!(op_lol,ad) else gen!(op_loe,ad)
918     end;
920   procedure pop(local:boolean; ad:integer; sz:integer);
921   begin assert not odd(sz);
922     if sz=wordsize then
923       begin if local then gen!(op_lal,ad) else gen!(op_lae,ad);
924         gen!(op_sti,sz)
925       end
926     else
927       if local then gen!(op_lol,ad) else gen!(op_loe,ad)
928     end;
930   procedure lexical(m:byte; lv:integer; ad:integer; sz:integer);
931   begin gen!(op_lex,level-lv); gen!(op_mdi,ad); gen!(m,sz) end;
933   procedure loadpos(var p:position; sz:integer);
934   begin with a do
935     if lv<0 then
936       #ifdef SEGMENTS
937         if ag<0 then
938           begin gen!(op_lsa,ag); gen!(op_mdi,ad); gen!(op_loi,sz) end
939         else
940           #endif
941           push(global,ad,sz)
942         else
943           if lv=level then push(local,ad,sz) else
944             lexical(op_loi,lv,ad,sz);
945     end;
947   procedure decaddr(var p:position);
948   begin if p.lv=0 then gen!(op_lae,p.ad) else loadpos(p,ptrsize) end;
950   procedure loadaddr;
951   begin with a do begin
952     case ok of
```

```
953     fixed;
954     with pos do
955       if lv<0 then
956         #ifdef SEGMENTS
957           if ag<0 then
958             begin gen!(op_lsa,ag); gen!(op_mdi,ad) end
959           else
960             #endif
961             gen!(op_lae,ad)
962           else
963             if lv=level then gen!(op_lal,ad) else
964               begin gen!(op_lex,level-lv); gen!(op_mdi,ad) end;
965         pfixed:=
966         loadpos(pos,ptrsize);
967         ploaded:=
968         ;
969         indexed:=
970         gen!(op_sas);
971         end; [case]
972         ak:=ploaded;
973     end end;
975   procedure load;
976   var sz:integer;
977   begin with a do begin
978     sz:=sizeof(asp); if not packbit then sz:=seven(sz);
979     if asp=all then
980       case ok of
981         cst:
982           gen!(op_loc,pos,ad); [only one-word scalars]
983         fixed:
984           loadpos(pos,sz);
985         pfixed:
986           begin loadpos(pos,ptrsize); gen!(op_loi,sz) end;
987         loaded:
988         ;
989         ploaded:
990           gen!(op_loi,sz);
991         indexed:
992           gen!(op_lsa);
993         end; [case]
994         ak:=loaded;
995     end end;
997   procedure store;
998   var sz:integer;
999   begin with a do begin
1000     sz:=sizeof(asp); if not packbit then sz:=seven(sz);
1001     if asp=all then
1002       case ok of
1003         fixed:
1004           with pos do
1005             if lv<0 then
1006               #ifdef SEGMENTS
1007                 if ag<0 then
1008                   begin gen!(op_lsa,ag);
```

```
1009     gen!(op_mdi,ad); gen!(op_sti,sz)
1010   end
1011   else
1012     #endif
1013     pop(global,ad,sz)
1014   else
1015     if level=lv then pop(local,ad,sz) else
1016       lexical(op_sti,lv,ad,sz);
1017   pfixed:=
1018   loadpos(pos,ptrsize); gen!(op_sti,sz) end;
1019   ploaded:=
1020   gen!(op_sti,sz);
1021   indexed:=
1022   gen!(op_sas);
1023   end; [case]
1024 end end;
1026   procedure fieldadr(off:integer);
1027   begin with a do
1028     if (ak=pfixed) and not packbit then pos.ad:=pos.ad+off else
1029       begin loadaddr; gen!(op_mdi,off) end
1030   end;
1032   procedure loadheap;
1033   begin if forsof(s.asp,[arrays..records]) then loadaddr else load end;
1035   [.....]
1037   procedure nextoh;
1038   begin
1039     col:=col+input; read(input,oh); e.ohno:=e.ohno+1; ehay:=col+oh;
1040   end;
1042   procedure nextin;
1043   begin
1044     if eof(input) then
1045       begin
1046         if not eofexpected then error(+03) else
1047           begin
1048             if f1used then begin gen!(ps_esc,seeffloat); genend end;
1049             gen!(ps_eof)
1050           end;
1051     #ifdef STANDARD
1052     goto 3999
1053   #endif
1054   #ifdef STANDARD
1055   halt
1056   #endif
1057   end;
1058   e.ohno:=0; e.lino:=e.lino+1; e.linar:=e.linar+1;
1059   if not including then
1060     begin e.orig:=e.orig; gvaline:=true end;
1061   end;
1063   procedure options(normal:boolean);
1064   var o:integer; i:integer;
```

```
1066   procedure goto;
1067   var b:byte;
1068   begin
1069     if normal then
1070       begin nextoh; o:=oh end
1071     else
1072       begin read(ah,b); c:=chr(b) end
1073   end;
1075   begin
1076     repeat goto;
1077     if (o='a') and (c='a') then
1078       begin ci:=o; goto i:=0;
1079       if c='.' then begin i:=1; goto end else
1080         if c='-' then goto else
1081           if c[0]<digit then
1082             repeat i:=i*10 + ord(o) - ord('0'); goto;
1083             until not c[0]<digit
1084           else i:=1;
1085           if i>=0 then
1086             if not normal then
1087               begin forceopt[ci]:=true; opt[ci]:=i end
1088             else
1089               if not forceopt[ci] then opt[ci]:=i;
1090           end;
1091           until c='.';
1092   end;
1094   procedure linedirective;
1095   var i,j:integer;
1096   begin i:=0; j:=0;
1097     repeat nextoh until (ch=' ') or eof;
1098     while ch<digit do
1099       begin i:=i*10 + ord(ch) - ord('0'); nextoh end;
1100     while (ch=' ') and not eof do nextoh;
1101     if (ch='*') or (eof) then error(+04) else
1102       begin nextoh;
1103         while (ch='*') and not eof do
1104           begin
1105             if ch='/' then j:=0 else
1106               begin if j=0 then e.fnam:=emptyfnam;
1107                 i:=i+1; if j<fnum then e.fnam[j]:=ch;
1108                 end;
1109             nextoh
1110           end;
1111       if source=emptyfnam then source:=e.fnam;
1112       including:=source<='o.fnam;
1113       i:=i-1; e.linar:=i;
1114       if not including then e.orig:=i
1115     end;
1116     while not eof do nextoh;
1117   end;
1119   procedure putdig;
1120   begin ix:=ix+1; if ix<max then strbuff[ix]:=ch; nextoh end;
```

```

1122 procedure indent;
1123 label 1;
1124 var i:integer;
1125 begin i:=0; id:=space;
1126 repeat
1127   if ch>upper then oh:=oh+(ord(oh)-ord('A')+ord('a'));
1128   if k<idmax then begin k:=k+1; id[k]:=ch end;
1129   match
1130   until oh<digit;
1131   [lower<0,upper<1,digit<2, ugly but fast]
1132   for i:=fru(k)-1 to fru(k)-1 do
1133     if ru[i]<id then
1134       begin sy:=ray(i); goto 1 end;
1135   sy:=idmax;
1136 1:
1137 end;

1139 procedure innumber;
1140 label 1;
1141 const lam = 10;
1142 var
1143   i:integer;
1144   is:=packed array[1..lam] of char;
1145 begin i:=0; sy:=idmax; val:=0;
1146 repeat putdig until oh<digit;
1147   if (oh='.') or (oh='e') or (oh='E') then
1148     begin
1149       if oh='.' then
1150         begin putdig;
1151           if oh='.' then
1152             begin seconddot:=true; ix:=ix-1; goto 1 end;
1153           if oh<digit then error(+05) else
1154             repeat putdig until oh<digit;
1155         end;
1156       if (oh='e') or (oh='E') then
1157         begin putdig;
1158           if (oh='e') or (oh='E') then putdig;
1159           if oh<digit then error(+06) else
1160             repeat putdig until oh<digit;
1161         end;
1162       if i>lam then begin error(+07); ix:=ixmax end;
1163       sy:=wordok; fl:=ord('r'); d:=ord('l'); val:=idbno;
1164       genidb(dibno); gen0(pe_rm); write(am,sp,room,ix);
1165       for i:=1 to ix do write(am,ord(strbuf[i])); genend;
1166     end;
1167   !:if (ch>lower) or (ch>upper) then teststandard;
1168   if sy=idmax then
1169     if i>lam then error(+08) else
1170       begin ix:=000000000; i:=ixmax+1;
1171         while i>0 do
1172           begin i:=i-1; if i:=strbuf[ix]; ix:=ix-1 end;
1173           if i:=comstr then
1174             while i:=com do
1175               begin val:=val*10 - ord('0') + ord(in[i]); i:=i-1 end
1176             else if i:=comlongstr then (dopt:=off) then
1177               begin sy:=longest; d:=ord('l'); val:=idbno;
1178             end;

```

```

1177   genidb(dibno); gen0(pe_con); write(am,sp,room,ixmax+1-1);
1178   while i:=com do
1179     begin write(am,ord(in[i])); i:=i-1 end;
1180   genend
1181   and
1182   error(+09)
1183   end
1184 end;

1186 procedure instrng(q:char);
1187 var i:integer;
1188 begin ix:=0; zerostring:=q;
1189 repeat
1190   repeat nextch; ix:=ix+1; if ix<max then strbuf[ix]:=ch;
1191   until (oh<=q) or eol;
1192   if oh<q then nextch else error(+10);
1193   until oh<=q;
1194   if not zerostring then
1195     begin ix:=ix-1; if ix=0 then error(+011) end
1196   else
1197     begin strbuf[ix]:=chr(0); if opt=off then error(+012) end;
1198     if (ix=1) and not zerostring then
1199       begin sy:=chr(ord); val:=ord(strbuf[1]) end
1200     else
1201       begin sy:=stringent; dibno:=idbno+1; val:=idbno;
1202         if i:=max then begin error(+013); i:=ixmax end;
1203         genidb(dibno); gen0(pe_rm); write(am,sp,room,ix);
1204         for i:=1 to ix do write(am,ord(strbuf[i])); genend;
1205       end;
1206 end;

1208 procedure incomment;
1209 var stop:=char;
1210 begin nextch; stop:='';
1211   if oh='*' then options:=true;
1212   while (oh<'') and (oh<stop) do
1213     begin stop:=''; if oh='*' then stop:='';
1214     if eol then nextch;
1215     if eol then nextch;
1216   end;
1217   if oh<' ' then teststandard;
1218   nextch
1219 end;

1221 procedure insym;
1222 [read next basic symbol of source program and return its
1223  description in the global variables sy, op, id, val and ix]
1224 label 1;
1225 begin
1226   !:same ch> of
1227   tabch;
1228   begin e:=ohno+e.ohno - e.ohno mod 8 + 8; nextch; goto 1 end;
1229   layout;
1230   begin if eol then nextch; nextch; goto 1 end;
1231   lower,upper:=idmax;
1232   digit:=innumber;

```

```

1233 quotech,dquotech;
1234 instrng(oh);
1235 colonch;
1236 begin nextch;
1237   if oh=':' then begin sy:=colon; nextch end else sy:=colon1;
1238 end;
1239 periodch;
1240 begin nextch;
1241   if seconddot then begin seconddot:=false; sy:=colon2 end else
1242     if oh='.' then begin sy:=colon2; nextch end else sy:=period;
1243 end;
1244 lessch;
1245 begin nextch;
1246   if oh='<' then begin sy:=less; nextch end else
1247     if oh='>' then begin sy:=less; nextch end else sy:=ltgt;
1248 end;
1249 greaterch;
1250 begin nextch;
1251   if oh='>' then begin sy:=less; nextch end else sy:=ltgt;
1252 end;
1253 lparench;
1254 begin nextch;
1255   if oh='(' then sy:=lparen else
1256     begin teststandard; incomment; goto 1 end;
1257 end;
1258 lbrackch;
1259 begin incomment; goto 1 end;
1260 rparench,lbrackch,rbrackch,comma,semicolon,arrowch,
1261 plusch,mulch,slash,star,equal;
1262 begin sy:=exp(ch); nextch end;
1263 otherch;
1264 begin
1265   if (oh='@') and (e.ohno=1) then llineinactive else
1266     begin error(+015); nextch end;
1267   goto 1
1268 end
1269 end (case)
1270 end;

1272 procedure nextf(f:symbol; err:integer);
1273 begin if sy=f then insym else error(-err) end;

1275 function find1(sy1,sy2:sos; err:integer):boolean;
1276 [symbol of sy1 expected. return true if sy in sy1]
1277 begin
1278   if not (sy in sy1) then
1279     begin error(-err); while not (sy in sy1+sy2) do insym end;
1280   find1:=sy in sy1;
1281 end;

1283 function find2(sy1,sy2:sos; err:integer):boolean;
1284 [symbol of sy1+sy2 expected. return true if sy in sy1]
1285 begin
1286   if not (sy in sy1+sy2) then
1287     begin error(-err); repeat insym until sy in sy1+sy2 end;
1288   find2:=sy in sy1;
1289 end;

```

```

1289 end;

1291 function find3(sy1:symbol; sy2:sos; err:integer):boolean;
1292 [symbol sy1 or one of sy2 expected. return true if sy found and skip]
1293 begin find3:=true;
1294   if not (sy in [sy1]+sy2) then
1295     begin error(-err); repeat insym until sy in [sy1]+sy2 end;
1296   if sy=sy1 then insym else find3:=false;
1297 end;

1299 function endofloop(sy1,sy2:sos; sy:symbol; err:integer):boolean;
1300 begin endofloop:=false;
1301   if find2(sy2+[sy1],sy1,err) then nextf(sy,err+1)
1302   else endofloop:=true;
1303 end;

1305 function lastsemicolon(sy1,sy2:sos; err:integer):boolean;
1306 begin lastsemicolon:=true;
1307   if not endofloop(sy1,sy2,semicolon,err) then
1308     if find2(sy2,sy1,err+2) then lastsemicolon:=false;
1309 end;

1311 [.....]

1313 function searchid(fidals: setof id):ip;
1314 [search for current identifier symbol in the name table]
1315 label 1;
1316 var lip:ip; is:=idless;
1317 begin lastap:=stop;
1318   while lastap<=id do
1319     begin lip:=lastap+1;
1320     while lip<=id do
1321       if lip<=id then
1322         if lip<=id then
1323           begin
1324             if lip<=id then
1325               lip:=lip+1;
1326             goto 1
1327           end
1328         else lip:=lip*.link
1329         else
1330           if lip<=id then lip:=lip*.link else lip:=lip*.link;
1331         lastap:=lastap*.link;
1332       end;
1333   err:=id<016,id;
1334   if types in fidals then is:=types else
1335   if vars in fidals then is:=vars else
1336   if const in fidals then is:=const else
1337   if proc in fidals then is:=proc else
1338   if func in fidals then is:=func else is:=false;
1339   lip:=endoflip(is);
1340 1:
1341   searchid:=lip;
1342 end;

1344 function searchsection(fip: ip):ip;

```

```

1345 (to find record fields and forward declared procedure id's
1346 ->procedure pfdclaration
1347 ->procedure selector)
1348 label 1;
1349 begin
1350 while flp<=all do
1351   if flp.name=id then goto 1 else
1352   if flp.name=ec id then flp:=flp.rlink else flp:=flp.llink;
1353   searchsection:=flp
1354 end;
1355
1356 function searchlab(flpl:lp; val:integer):lp;
1357 label 1;
1358 begin
1359 while flp<=all do
1360   if flp.labval=val then goto 1 else flp:=flp.nextlp;
1361   searchlab:=flp
1362 end;
1363
1364 procedure oponent(t:twostrut);
1365 var op:integer;
1366 begin with a do begin
1367   case is of
1368   ir: begin op:=op_dif; asp:=realptr; fltused:=true end;
1369   ri: begin op:=op_ofi; asp:=intptr; fltused:=true end;
1370   il: begin op:=op_oid; asp:=longptr end;
1371   li: begin op:=op_odi; asp:=intptr end;
1372   lr: begin op:=op_ofd; asp:=realptr; fltused:=true end;
1373   rl: begin op:=op_ofd; asp:=longptr; fltused:=true end;
1374   end;
1375   gen0(op)
1376 end end;
1377
1378 procedure negste(l1:integer);
1379 var l2:integer;
1380 begin
1381   if a.asp=intr then gen0(op_neg) else
1382   begin l2:=l1; gen0(op_loo,0);
1383   if a.asp=longptr then
1384     begin oponent(l1); exchange(l1,l2); gen0(op_dab) end
1385   else (realptr)
1386     begin oponent(l1); exchange(l1,l2); gen0(op_fab) end
1387   end;
1388 end;
1389
1390 function desub(fsp:sp);
1391 begin
1392   if formof(fsp,subrange) then fsp:=fsp.rangetype; desub:=fsp
1393 end;
1394
1395 function nicescalar(fsp:sp):boolean;
1396 begin
1397   if fsp=ll then nicescalar:=true else
1398   nicescalar:=(fsp.for=scalar) and (fsp<=realptr) and (fsp<=longptr)
1399 end;

```

```

1457 function compat(p,q:sp):twostrut;
1458 begin compat:=noeq;
1459 if eqstrut(p,q) then compat:=eq else
1460   begin p:=desub(p); q:=desub(q);
1461   if eqstrut(p,q) then compat:=subeq else
1462   if p.for=q.for then
1463     case p.for of
1464     scalar:
1465       if (p=intr) and (q=realptr) then compat:=ir else
1466       if (p=realptr) and (q=intptr) then compat:=ri else
1467       if (p=intptr) and (q=longptr) then compat:=il else
1468       if (p=longptr) and (q=intptr) then compat:=li else
1469       if (p=longptr) and (q=realptr) then compat:=lr else
1470       if (p=realptr) and (q=longptr) then compat:=rl else
1471       ;
1472     pointer:
1473       if (p=nlptr) or (q=nlptr) then compat:=eq;
1474     power:
1475       if p=emptyset then compat:=se else
1476       if q=emptyset then compat:=se else
1477       if compat(p,elset,q,elset) <= subeq then
1478         if p.sflag=q.sflag then compat:=eq;
1479     arrays:
1480       if string(p) and string(q) and (p.size=q.size) then
1481         compat:=eq;
1482     files,array,records: ;
1483   end;
1484 end;
1485
1486
1487 procedure checkasp(fsp:sp; err:integer);
1488 var ts:twostrut;
1489 begin
1490   ts:=compat(a.asp,fsp);
1491   case ts of
1492   eq:
1493     if fsp<=all then if withfile in fsp.sflag then aspperr(err);
1494   subeq:
1495     checkbada(fsp);
1496   li:
1497     begin oponent(ts); checkasp(fsp,err) end;
1498   ll,rl,lr,lr:
1499     oponent(ts);
1500   eq:
1501     aspendemptysat(fsp);
1502   notaq,ri,se:
1503     aspperr(err);
1504   end;
1505 end;
1506
1507 procedure force(fsp:sp; err:integer);
1508 begin load; checkasp(fsp,err) end;
1509
1510 function neident(kl:ld; id:sp; nst:lp; err:integer):lp;
1511 begin neident:=null;
1512 if sp<=ident then error(err) else

```

```

1401 function bounds(fsp:sp; var fmin,fmax:integer):boolean;
1402 (compute bounds if possible, else return false)
1403 begin bounds:=false; fmin:=0; fmax:=0;
1404 if fsp<=all then
1405   if fsp.for= subrange then
1406     begin fmin:=fsp.min; fmax:=fsp.max; bounds:=true end else
1407   if fsp.for=scalar then
1408     if fsp.foomat<=0 then
1409       begin fmin:=0; fmax:=fsp.foomat.value; bounds:=true end
1410   end;
1411
1412 procedure genrok(fsp:sp);
1413 var min,max,sno:integer;
1414 begin
1415   if opt['r']<=off then if bounds(fsp,min,max) then
1416     begin
1417       if fsp.for=scalar then sno:=fsp.scalno else sno:=fsp.subrno;
1418       if sno=0 then
1419         begin dibo:=dibo+1; sno:=dibo;
1420         genib(dibo); genl(pe_rom,min); genot(max); genod;
1421         if fsp.for=scalar then fsp.scalno:=sno else
1422         fsp.subrno:=sno
1423         end;
1424       end(op_rok,sno);
1425     end;
1426 end;
1427
1428 procedure checkbda(fsp:sp);
1429 var min,max1,min2,max2:integer; bool:boolean;
1430 begin
1431   if bounds(fsp,min,max1) then
1432     begin bool:=bounds(a.asp,min2,max2);
1433     if (bool=false) or (min2<min1) or (max2>max1) then
1434       genrok(fsp);
1435     end;
1436   a.asp:=fsp;
1437 end;
1438
1439 function eqstrut(p,q:sp):boolean;
1440 begin eqstrut:=(p=q) or (p=ll) or (q=ll) end;
1441
1442 function string(fsp:sp):boolean;
1443 var l:sp;
1444 begin string:=false;
1445 if formof(fsp,array) then
1446   if eqstrut(fsp,eltype.charptr) then
1447     if speak in fsp.sflag then
1448       begin l:=fsp.inctype;
1449       if l=ll then string:=true else
1450       if lsp.for= subrange then
1451         if lsp.rangetype=intr then
1452           if lsp.min=1 then
1453             string:=true
1454         end;
1455 end;

```

```

1513   begin neident:=newip(kl.id,ld,nt); inay end
1514 end;
1515
1516 function stringstrut:sp;
1517 var l:sp;
1518 begin (only used when ix and zerostring are still valid)
1519 if zerostring then l:=stringptr else
1520   begin l:=newip(array,ifcharize); l.sflag:=lspeak;
1521   l.p.selttype:=charptr; l.inctype:=null;
1522   end;
1523 stringstrut:=l;
1524 end;
1525
1526 function address(var lc:integer; az:integer; pack:boolean):integer;
1527 begin
1528   if lc >= maxint-az then begin error(+017); lc:=0 end;
1529   if (not pack) or (az=1) then if odd(lc) then lc:=lc-1;
1530   address:=lc;
1531   lc:=lc+az;
1532 end;
1533
1534 function reserve(s:integer):integer;
1535 var r:integer;
1536 begin r:=address(b.lo,s,false); genreg(r,s,100); reserve:=r;
1537 if b.lo>max then lmax:=b.lo
1538 end;
1539
1540 function arraysize(fsp:sp; pack:boolean):integer;
1541 var sz,min,max,tot,n:integer;
1542 begin sz:=sizeof(fsp.selttype);
1543 if not pack then sz:=sz*8;
1544 if bounds(fsp.inctype,min,max) then (we checked before)
1545   dibo:=dibo+1; fsp.arpos.lv:=0; fsp.arpos.ed:=dibo;
1546   genib(dibo); genl(pe_rom,min); genot(max-min);
1547   genot(max); genod;
1548   sz:=sz-min+1; tot:=sz*s;
1549   if sz<0 then if tot div sz < 0 then begin error(+018); tot:=0 end;
1550   arraysize:=tot
1551 end;
1552
1553 procedure treealk(flpl:lp);
1554 var l:lp; i:integer;
1555 begin
1556   if flp<=all then
1557     begin treealk(flpl.llink); treealk(flpl.rlink);
1558     if flp.kl=vars then
1559       begin if not (used in flp.iflag) then errid(-+019),flp.name;
1560       if not (assigned in flp.iflag) then errid(-+020),flp.name;
1561       l:=flp.idtype;
1562       if not (sorg in flp.iflag) then
1563         genreg(flpl.vpos.ed,ssizeof(lsp,ord(formof(lsp,pointer)))));
1564       if flp<=all then if withfile in l.sflag then
1565         if lsp.for=files then
1566           if level=1 then
1567             begin
1568               for i:=2 to arg do with arg[i] do

```

```

1569 if name=fip.name then ad:=fip.vpos.ad
1570 else
1571   begin
1572     if not (refer in fip.iflag) then
1573       begin gen(op_wrt,0);
1574       gen(top_lal,fip.vpos.ad); genop(CLS)
1575       end
1576     end
1577   end
1578   if level<1 then errid:=(+021),fip.name
1579   end
1580 end;
1581 end;
1582 end;

1584 procedure constant(fays:soa; var fapisp; var fval:integer);
1585 var signed_min:boolean; lip:lip;
1586 begin signed:=(fayspluss) or (faysminy);
1587 if signed then begin min:=signed; insyn end else min:=false;
1588 if find((ident..pluss),fays,+022) then
1589   begin fval:=val;
1590     case of
1591       stringst: fap:=stringst;
1592       charst: fap:=charst;
1593       intst: fap:=intst;
1594       realst: fap:=realst;
1595       longest: fap:=longst;
1596       shortest: fap:=shortst;
1597       ident:
1598         begin lip:=searchid((konst));
1599         fap:=lip.idtype; fval:=lip.value;
1600         end
1601       end;
1602   end;
1603   if signed then
1604     if (fap<intst) and (fap<realst) and (fap<longst) then
1605       error(+023)
1606     else if min then fval:= -fval;
1607     (note: negating the v-number for reals and longs)
1608   insyn;
1609   end;
1610   end;
1611   end;
1612   function outinteger(fays:soa; fapisp; err:integer):integer;
1613   var laptyp: lval_min_max_integer;
1614   begin constant(fays,lap,lval);
1615   if fap<lap then
1616     if (errtype=dupst) then
1617       begin
1618         if bounds(fap,min_max) then
1619           if (lval<min) or (lval>max) then error(+024)
1620         end
1621       else
1622         begin error(err); lval:=0 end;
1623       outinteger:=lval
1624     end;

```

```

1626 [.....]
1628 function typid(arr:integer):sp;
1629 var lip:lip; lipisp;
1630 newsubrange:boolean;
1631 begin lip:=nil;
1632 if ay<ident then error(err) else
1633   begin lip:=searchid((types)); lip:=lip.idtype; insyn end;
1634 typid:=lip
1635 end;

1636 function simpletp(fays:soa):sp;
1637 var lip:lip; lipisp; lip,hip:lip; min,max:integer; lnpmp;
1638 newsubrange:boolean;
1639 begin lip:=nil;
1640 if find((ident..parent),fays,+025) then
1641   if ay<parent then
1642     begin insyn; lip:=top; [decl. const. local to innermost block]
1643     while top<>occur do top:=top.nlink;
1644     lip:=newsp((smallr,wordsize); hip:=nil; max:=0;
1645     repeat lip:=newident((konst,lip,hip,+026);
1646     if lip<nil then
1647       begin enterid(lip);
1648       hip:=lip; lip.value:=max; max:=max+1
1649     until endofloop(fays,(parent),(ident),comma,+027); (+028)
1650     if max<8 then lip.size:=bytesize;
1651     lip.foo:=shp; top:=lip; nextif((parent,+029);
1652     end
1653   else
1654     begin newsubrange:=true;
1655     if ay<ident then
1656       begin lip:=searchid((types,konst)); insyn;
1657       if lip.klasstype then
1658         begin lip:=lip.idtype; newsubrange:=false end
1659       else
1660         begin lip:=lip.idtype; min:=lip.value end
1661     end
1662   else constant(fays,(colon2,ident..pluss),lip,min);
1663   if newsubrange then
1664     begin lip:=newsp(subrange,wordsize); lip.subno:=0;
1665     if not minmaxr(lip) then
1666       begin error(+030); lip:=nil; min:=0 end;
1667     lip.rangetype:=lap;
1668     nextif((colon2,+031); max:=outinteger(fays,lip,+032);
1669     if min>max then begin error(+033); max:=min end;
1670     if (min<0) and (max<8) then lip.size:=bytesize;
1671     lip.min:=min; lip.max:=max
1672   end;
1673   end;
1674   simpletp:=lip
1675 end;

1676 function arraytp(fays:soa;
1677   atyp:atstructform;
1678   sflag:flagset;

```

```

1681 function element(fays:soa):sp
1682   :sp;
1683 var lip,lip1,hip:lip; min,max:integer; ok:boolean; sepy:symbol; lipisp;
1684 ok:=true;
1685 begin insyn; nextif(lbreak,+034); hip:=nil;
1686 repeat lip:=newsp(atyp,0); lip:=lip.arpos;
1687 lip.min:=hip; lip.max:=lip; (link reversed)
1688 if artype=array then
1689   begin sepy:=colon; ok:=ident;
1690   lip:=newident((currhd,lip,min,+035);
1691   if lip<nil then enterid(lip);
1692   nextif((colon2,+036);
1693   lip:=newident((currhd,lip,min,+037);
1694   if lip<nil then enterid(lip);
1695   nextif((colon2,+038); lip:=typid(+039);
1696   ok:=ok and (not (dupst(lip)));
1697   end
1698 else
1699   begin sepy:=comma; ok:=((ident..parent);
1700   lip:=simpletp(fays,(comma,"break,ofy,ident..packeddy));
1701   ok:=bounds(lip,min,max)
1702   end;
1703   if not ok then begin error(+040); lip:=nil end;
1704   lip.lntype:=lap
1705   until endofloop(fays,(break,ofy,ident..packeddy),ok,sepy,+041); (+042)
1706   nextif(lbreak,+043); nextif(ofy,+044);
1707   lip:=element(fays);
1708   if lip<nil then sflag:=sflag + lip.sflag + [withfile];
1709   repeat (reverse link and compute aim)
1710     lip:=lip.nlink;
1711     if artype=array then lip.size:=arraysize(lip,speak in sflag);
1712     lip:=lip; hip:=lip;
1713     until lip:=nil; (lip points to array with highest dimension)
1714   arraytp:=lip
1715   end;

1716 function typ(fays:soa):sp;
1717 var lip,lipisp; ok:=true; min_max:integer;
1718 sflag:=flagset; lipisp;

1719

1722 function fldlist(fays:soa):sp;
1723 (level 2: << typ)
1724 var lip,hip,lipisp; lipisp;

1725 function varpart(fays:soa):sp;
1726 (level 3: << fldlist << typ)
1727 var lip,lipisp; lip,headsp,hip,hipisp; lipisp;
1728 min_max:=int; var:=integer; lid:=alpha;
1729 begin insyn; lip:=nil; lipisp;
1730 lip:=newsp(lap,0);
1731 if ay<ident then error(+045) else
1732   begin lid:=id; insyn;
1733   if ay<colon then
1734     begin lip:=newsp(field,lid,nil,nil); enterid(lip); insyn;
1735     if ay<ident then error(+046) else

```

```

1736   begin lid:=id; insyn end;
1737 end;
1738 if ay<ofy then (otherwise you may destroy id)
1739   begin lid:=lid; lip:=searchid((types)) end;
1740 end;
1741 if lip:=nil then tfap:=nil else tfap:=lip.idtype;
1742 if bounds(tfap,int,var) then var:=var-int-1 else
1743   begin var:=0;
1744   if tfap<nil then begin error(+047); tfap:=nil end
1745 end;
1746 lip:=lip; tfap:=tfap;
1747 if tip<nil then (explicit tag)
1748   begin tip:=tfap;
1749   lip.foo:=address(of,sizeof(tfap),speak in sflag)
1750 end;
1751 nextif(ofy,+048); min:=0; max:=min; headsp:=nil;
1752 repeat hip:=nil; (for each caselabel list)
1753   repeat var:=var-1;
1754     int:=outinteger(fays,(ident..pluss,comma,colon1,parent,
1755     semicolon,comma,rparent),tfap,+049);
1756     lip:=headsp; (each label may occur only once)
1757     while lip<nil do
1758       begin if lip.varval<int then error(+050);
1759       lip:=lip.nlink
1760     end;
1761     var:=newsp(var,int,0); var.varval:=int;
1762     var.nlink:=headsp; headsp:=var; (chain of case labels)
1763     var.subst:=hip; hip:=var;
1764     (use this field to link labels with same variant)
1765   until endofloop(fays,(colon1,parent,semicolon,comma,rparent),
1766   (ident..pluss),comma,+051); (+052)
1767   nextif((colon1,+053); nextif((parent,+054);
1768   top:=fldlist(fays,(parent,semicolon,ident..pluss));
1769   if ok:=true then min:=0;
1770   while var<nil do
1771     begin var:=min; ok:=hip:=var.subst;
1772     var:=var.subst; top:=lip; hip:=hip
1773   end;
1774   nextif((parent,+055);
1775   ok:=min;
1776   until lastsemicolon(fays,(ident..pluss),+056); (+057 +058)
1777   if var>0 then error(+059);
1778   top.foo:=headsp; top.size:=min; ok:=min; varpart:=top;
1779   end;

1780 begin (fldlist)
1781   if find((ident),fays,(array),+060) then
1782     repeat lip:=nil; hip:=nil;
1783     repeat lip:=newident((field,nil,nil,+061);
1784     if lip<nil then
1785       begin enterid(lip);
1786       if lip:=nil then hip:=lip else lip:=lip.nlink; lip:=lip;
1787       end;
1788     until endofloop(fays,(colon1,ident..packeddy,semicolon,comma),
1789     (ident),comma,+062); (+063)
1790   nextif((colon1,+064);

```



