

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 Steyvenan 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 {!- : no source line numbers}
39 {%- : no subrange checking}
40 {%- : no assertion checking}
41 #ifdef STANDARD
42 {%- : test conformance to standard}
43 #endif
44
45 program pem(input,em1,errors);
46 { This Pascal compiler produces EM1 code as described in
47 - A.S.Tanenbaum, J.W.Stevenson & H. van Steyvenan,
48 "Description of an experimental machine architecture for use of
49 block structured languages" Informatica rapport 54,
50 - K.Jensen & S.Wirth, PASCAL user manual and report, Springer-Verlag.
51 Several options may be given in the normal pascal way. Moreover,
52 a positive number may be used instead of + and -. The options are:
53 a: interpret assertions (+)
54 o: C-type strings allowed (-)
55 e: type long may be used (-)
56 }
```

```
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 booleanize = bytezize;
119 charsize = bytezize;
120 intsize = shortzize;
121 buffsize = 512;
122 maxsetsize = 4096; [t15 div bytezite]
123
124 {maximal indices}
125 lmax = 8;
126 fmax = 14;
127 smax = 72;
128 rmax = 72;
129 lmax = 10;
130
131 {opt values}
132 off = 0;
133 on = 1;
134
135 {for push and pop;}
136 global = false;
137 local = true;
138
139 {set bounds}
140 misetint = 0;
141 maxsetint = 15; {default}
142
143 {constants describing the compact EM1 code}
144 MAGICLOW = 172;
145 MAGICHIGH = 0;
146 memerror = 0;
147 memopsize = 1;
148 memoptimal = 2;
149 memreg = 3;
150 memline = 4;
151 memfloats = 5;
152
153 {ASCII characters}
154 Lab = 9;
155 newline = 10;
156 abort = 11;
157 formfeed = 12;
158 crret = 13;
159
160 {miscellaneous}
161 memarg = 127; {maximal segment number}
162 memcharord = 127; {maximal ordinal number of chars}
163 memarg = 13; {maximal index in argv}
164 rlim = 34; {number of reserved words}
165 spoc = ;
166 emptyform = ;
167
168
```

```
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 wbit = 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 punysize = 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 ptrsize, realsize and floatsize 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 striped 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
```

```
169 type
170 {scalar types}
171 symbol = (comma,semicolon,colon,colon2,notary,lbrace,ident,
172 intout,charact,realout,longout,stringout,alicut,ainy,
173 plusy,percent,arrow,arrayy,recordy,setsy,filesy,
174 packedy,progy,labely,consty,typesy,varsy,procsy,
175 funcy,beginy,gotosy,ifsy,whiley,repeaty,foray,
176 withy,casety,becomes,staray,divy,mody,alseny,
177 anday,oray,exory,noty,asy,asy,asy,
178 leay,iny,andy,elasy,untilay,ofay,dasy,
179 downtoy,coy,thensy,rbrack,rparent,period
180 ); {the order is important}
181 chartype = (lower,upper,digit,layout,tabch,
182 quotech,quotech,ordtech,periodch,leasch,
183 greaterch,lparentch,lbracoch,
184 ); {different entries}
185 rparentch,lbrackch,rbrackch,ocommach,semich,arrowch,
186 plusch,minch,alash,star,equal,
187 ); {also symbols}
188 others
189 );
190
191 standpf = (pread,preadln,write,partials,pput,pgot,
192 preat,prewrite,pnes,dispose,ppack,punpack,
193 pmack,prelease,ppage,phalt,
194 ); {all procedures}
195 foof,fooln,foab,foqr,ford,fofr,fpred,foact,foadd,
196 furmo,found,fsin,foos,foexp,foqr,ftn,foctan
197 ); {all functions}
198 ); {the order is important}
199
200 libnames = (INL,IFL,CLS,VM ,
201 OFN,GETX,END,DBC,END,ML,ML,
202 ); {see input files}
203
204 CSE,FUTY,WEI,MSI,WC,WC,WC,WC,WC,WC,
205 WMB,WBR,WBR,WML,MSL,WMP,WRI,MSZ,WLM,PAC,
206 ); {see output files, order important}
207
208 ABS,END,SEN,COS,EXP,SQ,LOC,ATN
209 ); {floating point}
210
211 ABI,ABL,BCP,BYS,HEX,SAV,EST,IDI,HLT,
212 ASS,OTO,PAC,WMP,DLS,ARI,MDI,MDL,
213 ); {miscellaneous}
214
215
216 structform = (scalar,subrange,pointer,power,files,arrays,arrayy,
217 records,variant,tag); {order important}
218
219 structflag = (speak,withfile);
220
221 identflag = (refer,used,assigned,orag,assigned);
222
223 declsize = (types,font,vars,field,orand,proc,func);
224
225 kindofpr = (standard,formal,actual,extra,forward);
226
227 where = (blk,rec,word);
228
229 strkinds = (out,flrd,pflrd,loaded,ploded,indented);
230
231 tuostrunc = (eq,subeq,lr,rl,il,il,lr,rl,se,se,note);
232
233 {subrange types}
234 sgrange = 0..max;
235 idrange = 1..lmax;
236 frange = 1..fmax;
237
238
```

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 blk:pp; (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:ap; (type of bounds)

309 subno:integer; (number of subr descriptor)

310);

311 pointer : (altype:ap; (type of pointed object)

312 power : (alast:ap; (type of set elements)

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

314 arrays,orarray: (type of array elements)

315 inttype:ap; (type of array index)

316 arpos:position; (position of array descriptor)

317);

318 records : (f:fld:ip; (points to first field)

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

320);

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

322 mtrvar:ap; (next equilevel variant)

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

324);

325 tag : (f:atvar:ap; (first variant of case)

326 fldtag:ap; (type of tag)

327);

328 end;

329

331 identifier:record

332 idtype:ap; (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:standpf); (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 var (the most frequent used externals are declared first)

377 sy:symbol; (last symbol)

378 s:sttr; (type,access method,position,value of expr)

379 (returned by inqpr)

380 ch:sttr; (last character)

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

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

383 ix:integer; (string length)

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

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

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

387 (same counters)

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

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

390 lmax:integer; (deepest level of nesting of ls)

391 level:integer; (current static level)

393 ptrsize:integer;

394 realize:integer; (file header size)

395 rbase: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 lop: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 intypedec: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 fuptr:ip; (head of chain of forward reference pointers)

415 progr:ip; (program identifier)

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

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

418 lasttag: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:sttr; (sttr 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

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

434 (symbol for reserved words)

435

436 ca:array[chr] of chartype;

437 (ch:char of a character)

438

439 ca:array(r:par:stch, equal) of symbol;

440 (symbol for single character symbols)

441 lms:array[1..lmax] of ip;

442 (addresses of pascal library routines)

443

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

445 (different options)

446

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

448 (used in searchid)

449

450 wdef:ip:array[ideclass] of ip;

451 (used in searchid)

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

```
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,type,i);
495  for j:=1 to i do write(eml,ord(a[j]));
496  end;
498  procedure genmp(a:libname);
499  var i:integer;
500  begin gen0(op_cal); write(eml,sp_pnm,4);
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<ppos.lv<1 then n:=fil.p_name else
508  begin n:=fil.p_name; i:=1; j:=1; i:=fil.p_pno;
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 gen1(op_lin,e.orig)
521  end;
523  procedure genreg(ad,az,nr:integer);
524  begin
525  if az<wordsize then
526  begin gen1(sp_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 gen1(sp_mes,meserror); genend end
537  end;
539  procedure error(err:integer);
540  begin e:=sp_spaces; e:=e-1; puterr(err) end;
542  procedure errid(err:integer; var id:alpha);
543  begin e:=e+id; e:=e-1; puterr(err) end;
545  procedure errint(err:integer; i:integer);
546  begin e:=e+i; e:=sp_spaces; puterr(err) end;
548  procedure aspperr(err:integer);
549  begin if e.sp<nil then begin error(err); e.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 nam:alpha; lip,lip1:ip; lleft,again:boolean;
559  begin nam:=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=nam then
565  begin lip:=lip.p_llink; lleft:=true end
566  else
567  begin if lip.p_name=nam then again:=true; (name conflict)
568  lip:=lip.p_rlink; lleft:=false;
569  end;
570  until lip=nil;
571  if lleft then lip1:=lip else lip1:=lip.p_rlink:=fil;
572  end;
573  fil.p_llink:=nil; fil.p_rlink:=nil;
574  if again then errid(+e02,nam);
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:=fzid; 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,ovrbrd: (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.fctid:=actual;
606  initpos(p,p_pos); p.p.pno:=0; p.p.parhead:=nil; p.p.head:=0
607  end;
608  end;
609  p.p.name:=p.p.klass:=kl; p.p.idtype:=id; p.p.next:=ent;
610  p.p.llink:=nil; p.p.rlink:=nil; p.p.iflag:=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.dtype:=nil end;
623  power:
624  new(p,power);
625  files:
626  begin newp(files); sf:=[]; (with file) end;
627  arrays,ovrbrd: (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.sf:=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.occ:=ak:=ak; top.p.rlink:=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  rsy[0]:='ifay'; rsy[1]:='doay'; rsy[2]:='ofay';
660  rsy[3]:='toay'; rsy[4]:='inay'; rsy[5]:='oray';
661  rsy[6]:='enday'; rsy[7]:='foray'; rsy[8]:='nilost';
662  rsy[9]:='varay'; rsy[10]:='divay'; rsy[11]:='moday';
663  rsy[12]:='setay'; rsy[13]:='anday'; rsy[14]:='notay';
664  rsy[15]:='thenay'; rsy[16]:='elseay'; rsy[17]:='withay';
665  rsy[18]:='caseay'; rsy[19]:='typeay'; rsy[20]:='gotoy';
666  rsy[21]:='fileay'; rsy[22]:='beginay'; rsy[23]:='untilay';
667  rsy[24]:='whileay'; rsy[25]:='arrayay'; rsy[26]:='constay';
668  rsy[27]:='labelay'; rsy[28]:='repeatay'; rsy[29]:='recorday';
669  rsy[30]:='down toay'; rsy[31]:='packeday'; rsy[32]:='progray';
670  rsy[33]:='functay'; rsy[34]:='proceday';
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   fvw(5):=28; fvw(6):=32; fvw(7):=33; fvw(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))::=stabsch;
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('&#x2014;')::=longdash; os('&#x2018;')::=leftquot; os('&#x2019;')::=rightquot;
690   os('&#x201c;')::=leftdblquot; os('&#x201d;')::=rightdblquot;
691   os('&#x201e;')::=lefttriplequot; os('&#x201f;')::=righttriplequot;
692   os('&#x201a;')::=singleleftquot; os('&#x201b;')::=singlerightquot;
693   (single character symbols in char type order)
694   os('&#x2011;')::=dash; os('&#x2012;')::=longdash;
695   os('&#x2013;')::=dash; os('&#x2014;')::=longdash;
696   os('&#x2015;')::=dash; os('&#x2016;')::=longdash;
697   os('&#x2017;')::=dash; os('&#x2018;')::=leftquot;
698   os('&#x2019;')::=rightquot; os('&#x201c;')::=leftdblquot;
699   os('&#x201d;')::=rightdblquot; os('&#x201e;')::=lefttriplequot;
700   os('&#x201f;')::=righttriplequot;
701   procedure init3;
702   var p,q:ip; k:kdclass;
703   begin
704   (undefined identifier pointers used by searchid)
705   for k:types to fno do
706   underfp[k]::=newip(k,spaces.all,nil);
707   (standard type pointers, some size are filled in by handleopts)
708   intptr :=newip(scalar.intsize);
709   realptr :=newip(scalar.realsize);
710   longptr :=newip(scalar.longsize);
711   charptr :=newip(scalar.charsize);
712   boolptr :=newip(scalar.boolsize);
713   nilptr :=newip(pointer,0);
714   stringptr:=newip(pointer,0);
715   emptyset :=newip(power.intsize); emptyset^.elset:=nil;
716   textptr :=newip(files,0); textptr^.fltype:=charptr;
717   (standard type names)
718   enterid(newip(types,'integer','intptr,nil));
719   enterid(newip(types,'real','realptr,nil));
720   enterid(newip(types,'char','charptr,nil));
721   enterid(newip(types,'boolean','boolptr,nil));
722   enterid(newip(types,'text','textptr,nil));
723   (standard constant names)
724   q:=nil; p:=newip(const,'false','boolptr,q); enterid(p);
725   q:=p; p:=newip(const,'true','boolptr,q); enterid(p);
726   boolptr^.foast:=p;
727   p:=newip(const,'maxint','intptr,nil); p^.value:=maxint; enterid(p);
728   p:=newip(const,'spaces,chars,all); p^.value:=maxcharord;

```

```

729   charptr^.foast:=p;
730   end;
731
732   procedure init3;
733   var j:standpf; p:ip; q:mp;
734   p:=array(standpf) of alpha;
735   ftype:=array(foef..faroten) of sp;
736   begin
737   (names of standard procedures/functions)
738   p[read]:=read; p[readin]:=readin;
739   p[write]:=write; p[writein]:=writein;
740   p[put]:=put; p[putin]:=putin;
741   p[page]:=page; p[pgot]:=pgot;
742   p[ppage]:=ppage; p[ppreset]:=ppreset;
743   p[ppwrite]:=ppwrite; p[ppow]:=ppow;
744   p[pprepare]:=pprepare; p[ppack]:=ppack;
745   p[pplease]:=pplease; p[ppack]:=ppack;
746   p[ppor]:=ppor; p[ppalt]:=ppalt;
747   p[ppabs]:=ppabs; p[ppord]:=ppord;
748   p[ppord]:=ppord; p[ppord]:=ppord;
749   p[ppord]:=ppord; p[ppord]:=ppord;
750   p[ppord]:=ppord; p[ppord]:=ppord;
751   p[ppord]:=ppord; p[ppord]:=ppord;
752   p[ppord]:=ppord; p[ppord]:=ppord;
753   p[ppord]:=ppord; p[ppord]:=ppord;
754   p[ppord]:=ppord; p[ppord]:=ppord;
755   p[ppord]:=ppord; p[ppord]:=ppord;
756   (parameter types of standard functions)
757   ftype[foef]:=nil; ftype[foein]:=nil;
758   ftype[foaf]:=nil; ftype[foaf]:=nil;
759   ftype[foaf]:=nil; ftype[foaf]:=nil;
760   ftype[foaf]:=nil; ftype[foaf]:=nil;
761   ftype[foaf]:=nil; ftype[foaf]:=nil;
762   ftype[foaf]:=nil; ftype[foaf]:=nil;
763   ftype[foaf]:=nil; ftype[foaf]:=nil;
764   ftype[foaf]:=nil; ftype[foaf]:=nil;
765   ftype[foaf]:=nil; ftype[foaf]:=nil;
766   (standard procedure/function identifiers)
767   for j:=read to p[read] do
768   begin new(p,proc,standpf); p^.klass:=proc;
769   p^.name:=p[j]; p^.pkind:=standpf; p^.key:=j; enterid(p);
770   end;
771   for j:=foef to faroten do
772   begin new(p,func,standpf); p^.klass:=func; p^.idtype:=ftype[j];
773   (idtype is used not for result type but for parameter type if)
774   p^.name:=p[j]; p^.pkind:=standpf; p^.key:=j; enterid(p);
775   end;
776   (program identifier)
777   prog:=newip(proc,'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   lnn[ELN]:=e; lnn[EFL]:=e; lnn[CLS]:=c;
786   lnn[VM]:=v;
787   lnn[OPW]:=o; lnn[GRX]:=g; lnn[ROI]:=r;
788   lnn[RDC]:=r; lnn[RDE]:=r; lnn[RDL]:=r;
789   lnn[RLM]:=r;
790   lnn[CMX]:=c; lnn[PTX]:=p; lnn[WRU]:=w;
791   lnn[MSI]:=m; lnn[MSU]:=m; lnn[MSC]:=m;
792   lnn[MSB]:=m; lnn[MSW]:=m; lnn[MSV]:=m;
793   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
794   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
795   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
796   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
797   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
798   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
799   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
800   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
801   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
802   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
803   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
804   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
805   lnn[MSL]:=m; lnn[MSL]:=m; lnn[MSL]:=m;
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+1;
811   opt['l']:=0;
812   opt['p']:=0;
813   opt['s']:=addresssize div wordsize; (default pointer size in words)
814   opt['t']:=0;
815   opt:=off;
816   (scalar variables)
817   b:=nil;
818   b:=nil;
819   b:=nil;
820   b:=nil;
821   b:=nil;
822   e:=nil;
823   e:=nil;
824   e:=nil;
825   e:=nil;
826   e:=nil;
827   e:=nil;
828   l:=0;
829   d:=0;
830   w:=0;
831   l:=0;
832   g:=0;
833   l:=0;
834   w:=0;
835   l:=0;
836   l:=0;
837   l:=0;
838   l:=0;
839   l:=0;
840   l:=0;

```

```

841   argv[0].ed:=1;
842   end;
843
844   procedure handleopts;
845   begin
846   opt:=opt;
847   opt:=opt;
848   opt:=opt;
849   opt:=opt;
850   realsize:=opt['r']*wordsize; realptr^.size:=realize;
851   ptrsize:=opt['p']*wordsize; nilptr^.size:=ptrsize;
852   fsize:=d+intsize+2*ptrsize;
853   textptr^.size:=fsize+bufsize; stringptr^.size:=ptrsize;
854   if opt<off then begin opt:=off; dopt:=off end;
855   else if opt['u']<off then os(' ');
856   if opt<off then enterid(newip(types,'string','stringptr,nil));
857   if opt<off then enterid(newip(types,'long','longptr,nil));
858   if opt['o']<off then begin gen((p,mes,mesoptoff); genend end;
859   if ptrsize<wordsize then begin gen((p,mes,mesvirtual); genend end;
860   if dopt<off then ftype:=true; (temporary kludge)
861   end;
862
863   (=====)
864
865   procedure trace(tname:alpha; fip:ip; var nandib:integer);
866   var i:integer;
867   begin
868   if opt['t']<off then
869   begin
870   if nandib=0 then
871   begin dibo:=dibo+1; nandib:=dibo; genld(dibo);
872   gen0(p,rum); write(m,sp_soon,8);
873   for i:=1 to 8 do write(m,ord(fip^.name[i])); genend;
874   end;
875   gen((op_wrt,0); gen0(op,mes,nandib);
876   gen0(op_out); genid(m,sp_pam,tname);
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 forms end;
882
883   function sizeof(fsp:sp):integer;
884   var s:integer;
885   begin s:=0;
886   if fsp<off then s:=fsp^.size;
887   if s<0 then if odd(s) then s:=s+1;
888   sizeof:=s;
889   end;
890
891   function even(i:integer):integer;
892   begin if odd(i) then i:=i+1; even:=i end;
893
894   procedure exchange(i1,i2:integer);
895   var d1,d2:integer;
896   begin d1:=i2-1; d2:=i1o-12;

```

```
897   if (d1<0) and (d2<0) then
898     begin gen!(ps_esc,d1); gen!(d2) end
899   end;
900
901   procedure setop(s:byte);
902   begin gen!(s,even(sizeof(s.asp))) end;
903
904   procedure s2pemptyset(fap:sp);
905   var i:integer;
906   begin
907     for i:=2 to sizeof(fap) div wordsize do gen!(op_loc,0); a.asp:=fap
908   end;
909
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;
919
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;
929
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;
932
933   procedure loadpos(var p:position; sz:integer);
934   begin with a do
935     if lv<0 then
936       #ifdef SECTIONS
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;
946
947   procedure deaddr(var p:position);
948   begin if p.lv=0 then gen!(op_lae,p.ad) else loadpos(p,ptrsize) end;
949
950   procedure loadadr;
951   begin with a do begin
952     case of

```

```
953     fixed;
954     with pos do
955       if lv<0 then
956         #ifdef SECTIONS
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         loadpos(pos,ptrsize);
966         ploaded;
967         ;
968         indexed;
969         gen!(op_sas);
970         end; [case]
971         sk:=ploaded;
972         end end;
973
974   procedure load;
975   var sz:integer;
976   begin with a do begin
977     sz:=sizeof(asp); if not packbit then sz:=seven(sz);
978     if asp=all then
979       case of
980         case of
981           case of
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             ploaded;
989             gen!(op_loi,sz);
990             indexed;
991             gen!(op_lsa);
992             end; [case]
993             sk:=loaded;
994             end end;
995
996   procedure store;
997   var sz:integer;
998   begin with a do begin
999     sz:=sizeof(asp); if not packbit then sz:=seven(sz);
1000     if asp=all then
1001       case of
1002         fixed;
1003         with pos do
1004           if lv<0 then
1005             #ifdef SECTIONS
1006               if ag<0 then
1007                 begin gen!(op_lsa,ag);
1008

```

```
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;
1025
1026   procedure fieldadr(off:integer);
1027   begin with a do
1028     if (sk=fixed) and not packbit then pos.ad:=pos.ad+off else
1029       begin loadadr; gen!(op_mdi,off) end
1030   end;
1031
1032   procedure loadheap;
1033   begin if forsof(s.asp,[arrays..records]) then loadadr else load end;
1034   [.....]
1035
1036   procedure nextch;
1037   begin
1038     col:=col+1; read(input,cb); e.ohno:=e.ohno+1; ehay:=col;
1039   end;
1040
1041   procedure nextln;
1042   begin
1043     if eof(input) then
1044       begin
1045         if not eof(input) then error(+03) else
1046           begin
1047             if fixed then begin gen!(ps_esc,seefloat); gen!(d) end;
1048             gen!(ps_esc)
1049           end;
1050     #ifdef STANDARD
1051     goto 3999
1052   #endif
1053   #ifdef STANDARD
1054   halt
1055   #endif
1056   end;
1057   e.ohno:=0; e.lino:=e.lino+1; e.linc:=e.linc+1;
1058   if not including then
1059     begin e.orig:=e.orig; gvaline:=true end;
1060   end;
1061
1062   procedure options(normal:boolean);
1063   var o:integer; i:integer;

```

```
1064
1065   procedure goto;
1066   var b:byte;
1067   begin
1068     if normal then
1069       begin nextch; o:=oh end
1070     else
1071       begin read(am,b); c:=chr(b) end
1072     end;
1073
1074   begin
1075     repeat goto;
1076     if (o='a') and (c='a') then
1077       begin ci:=o; goto i:=0;
1078       if c='.' then begin i:=1; goto end else
1079         if c='-' then goto else
1080           if c[0]<digit then
1081             repeat i:=i*10 + ord(o) - ord('0'); goto;
1082             until not c[0]<digit
1083           else i:=1;
1084           if i>=0 then
1085             if not normal then
1086               begin forceopt[ci]:=true; opt[ci]:=i end
1087             else
1088               if not forceopt[ci] then opt[ci]:=i;
1089             end;
1090           until c='.';
1091           end;
1092
1093   procedure linedirective;
1094   var i,j:integer;
1095   begin i:=0; j:=0;
1096     repeat nextch until (ch=' ') or eof;
1097     while chy=digit do
1098       begin i:=i*10 + ord(ch) - ord('0'); nextch end;
1099     while (ch=' ') and not eof do nextch;
1100     if (ch='*') or (eof) then error(+04) else
1101       begin nextch;
1102         while (ch='*') and not eof do
1103           begin
1104             if ch='/' then j:=0 else
1105               begin if j=0 then e.fname:=emptyfsm;
1106                 i:=i+1; if j<fmax then e.fname[j]:=ch;
1107                 end;
1108               nextch
1109             end;
1110             if source=emptyfsm then source:=e.fname;
1111             including:=source<=e.fname;
1112             i:=i-1; e.linc:=i;
1113             if not including then e.orig:=i
1114           end;
1115         while not eof do nextch;
1116       end;
1117
1118   procedure putdig;
1119   begin i:=i+1; if i<fmax then strbuff[i]:=ch; nextch end;

```

```

1122 procedure indent;
1123 label 1;
1124 var i:integer;
1125 begin i:=0; id:=space;
1126 repeat
1127   if ch>upper then ch:=chr(ord(ch)-ord('A')+ord('a'));
1128   if k<idmax then begin k:=k+1; id[k]:=ch end;
1129   nextch
1130 until ch=y<digit;
1131 [lower=0,upper=1,digit=2, ugly but fast]
1132 for i:=frk[i]-1 to frk[i]-1 do
1133   if rw[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 ix:=0; sy:=idmax; val:=0;
1146   repeat putdig until ch=y<digit;
1147   if (ch='.') or (ch='e') or (ch='E') then
1148     begin
1149       if ch='.' then
1150         begin putdig;
1151           if ch='.' then
1152             begin seconddot:=true; ix:=ix+1; goto 1 end;
1153           if ch=y<digit then error(+05) else
1154             repeat putdig until ch=y<digit;
1155         end;
1156       if (ch='e') or (ch='E') then
1157         begin putdig;
1158           if (ch='e') or (ch='E') then putdig;
1159           if ch=y<digit then error(+06) else
1160             repeat putdig until ch=y<digit;
1161         end;
1162       if ix>lam then begin error(+07); ix:=lam end;
1163       sy:=nextch; if second:=true; d:=no; m:=no; val:=idbno;
1164       genidb(dibno); genidp(pe_rm); write(am,sp,room,ix);
1165       for i:=1 to ix do write(am,ord(strbuf[i])); genid;
1166     end;
1167   !:if (ch=y<lower) or (ch=y<upper) then teststandard;
1168   if sy=idmax then
1169     if ix>lam then error(+08) else
1170       begin ix:=0; id:=space; i:=lam+1;
1171         while ix>0 do
1172           begin i:=i-1; id[i]:=strbuf[ix]; ix:=ix-1 end;
1173         if ix<maxinstr then
1174           while ix<max do
1175             begin val:=val*10 + ord('0') + ord(id[i]); i:=i+1 end
1176         else if (ix<maxlongstr) and (dopt<off) then
1177           begin sy:=longest; d:=no; m:=no; val:=idbno;
1178         end;

```

```

1177   genidb(dibno); genidp(pe_rm); write(am,sp,room,ix+1-1);
1178   while ix<lam do
1179     begin write(am,ord(id[i])); i:=i+1 end;
1180   genid
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 (ch=q) or eol;
1192   if ch=q then nextch else error(+10);
1193   until ch<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:=dibno+1; val:=dibno;
1202         if ix>max then begin error(+013); ix:=max end;
1203         genidb(dibno); genidp(pe_rm); write(am,sp,room,ix);
1204         for i:=1 to ix do write(am,ord(strbuf[i])); genid;
1205       end;
1206 end;

1208 procedure incomment;
1209 var stop:=char;
1210 begin nextch; stop:='';
1211 if ch='*' then options:=true;
1212 while (ch<'*') and (ch<stop) do
1213   begin stop:=''; if ch='*' then stop:='';
1214   if eol then nextch;
1215   if eol then nextch;
1216   end;
1217 if ch<'*' 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:=chno:=no.chno - e.chno 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(ch);
1235 colonch;
1236 begin nextch;
1237   if ch=':' 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 ch='.' then begin sy:=colon2; nextch end else sy:=period;
1243 end;
1244 lessch;
1245 begin nextch;
1246   if ch='<' then begin sy:=less; nextch end else
1247   if ch='>' then begin sy:=less; nextch end else sy:=ltgt;
1248 end;
1249 greaterch;
1250 begin nextch;
1251   if ch='>' then begin sy:=less; nextch end else sy:=ltgt;
1252 end;
1253 lparench;
1254 begin nextch;
1255   if ch='(' 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,minuch,slash,star,equal;
1262 begin sy:=exp(chay); nextch end;
1263 otherch;
1264 begin
1265   if (ch='@') and (e.chno=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 inps 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 inps 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 inps 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 inps until sy in [sy1]+sy2 end;
1296 if sy=sy1 then inps 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: setofid):ip;
1314 [search for current identifier symbol in the name table]
1315 label 1;
1316 var lip:ip; is:ideless;
1317 begin lastap:=stop;
1318 while lastap<nil do
1319   begin lip:=lastap^.fname;
1320   while lip<nil do
1321     if lip^.nameid then
1322       if lip^.kname in fidals then
1323         begin
1324           if lip^.kname+vs then if lip^.vpos.lv<level then
1325             lip:=lip^.lftag+lip^.lftag+more;
1326           goto 1
1327         end
1328       else lip:=lip^.rlink
1329     else
1330       if lip^.nameid then lip:=lip^.rlink else lip:=lip^.llink;
1331   lastap:=lastap^.llink;
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(flplp: 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=ntpr 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;

```

```

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 checkbnds(fsp:sp);
1429 var min,max1,min2,max2:integer; bool:boolean;
1430 begin
1431   if bounds(fsp,min,max1) then
1432     begin bool:=bounds(e.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,asetype.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=ntpr then
1452           if lsp.min=1 then
1453             string:=true
1454           end;
1455 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     (p=ntpr) and (q=realptr) then compat:=ir else
1465     (p=realptr) and (q=intptr) then compat:=ri else
1466     (p=intptr) and (q=longptr) then compat:=il else
1467     (p=longptr) and (q=intptr) then compat:=li else
1468     (p=longptr) and (q=realptr) then compat:=lr else
1469     (p=realptr) and (q=longptr) then compat:=rl else
1470     ;
1471     pointer:
1472     if (p=ntpr) or (q=ntpr) then compat:=eq;
1473     power:
1474     if p=emptyset then compat:=set else
1475     if q=emptyset then compat:=set else
1476     if compat(p,elset,q,elset) <= subeq then
1477       if p.sflag=q.sflag then compat:=eq;
1478     arrays:
1479     if string(p) and string(q) and (p.size=q.size) then
1480       compat:=eq;
1481     files,array,records: ;
1482     end;
1483   end;
1484 end;
1485
1486 procedure checkasp(fsp:sp; err:integer);
1487 var ts:twostrut;
1488 begin
1489   ts:=compart(a.asp,fsp);
1490   case ts of
1491   eq:
1492     if fsp<=all then if withfile in fsp.sflag then aspperr(err);
1493   subeq:
1494     checkbnds(fsp);
1495   il:
1496     begin oponent(ts); checkasp(fsp,err) end;
1497   lr,rl,lr,lr:
1498     oponent(ts);
1499   end;
1500   aspendemptysset(fsp);
1501   notaq,ri,ae:
1502     aspperr(err);
1503   end;
1504 end;
1505
1506 procedure force(fsp:sp; err:integer);
1507 begin load; checkasp(fsp,err) end;
1508
1509 function neident(kl:ld; id:sp; nst:ptr; err:integer):lp;
1510 begin neident:=null;
1511 if sp<=ident then error(err) else
1512

```

```

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.seldtype:=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:=asetof(fsp,asetype);
1543 if not pack then sz:=even(sz);
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   a:=max-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(flplp);
1554 var l:sp; i:integer;
1555 begin
1556   if flp<=all then
1557     begin treealk(flplp.llink); treealk(flplp.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(flplp.vpos.ed,asetof(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 argo do with argo[i] do

```

```

1569         if name=fip.name then ad:=fip.vpos.ad
1570     else
1571     begin
1572     begin
1573     if not (refer in fip.iflag) then
1574     begin gen(op,wrt,0);
1575     gen(top_lal,fip.vpos.ad); genop(CLS)
1576     end
1577     end
1578     else
1579     if level<1 then errid:=(+021),fip.name)
1580     end
1581 end;
1582
1583
1584 procedure constant(fays:soa; var fapisp; var fval:integer);
1585 var signed_min:boolean; lip:lip;
1586 begin signed:=(faypluss) or (fayminus);
1587 if signed then begin min:=ayminus; insyn end else min:=false;
1588 if find((ident..pluss),fays,+022) then
1589 begin fval:=val;
1590 case ay of
1591 stringst: fap:=stringst;
1592 charst: fap:=charst;
1593 intst: fap:=intptr;
1594 realst: fap:=realptr;
1595 longest: fap:=longptr;
1596 shortest: fap:=shortptr;
1597 ident:
1598 begin lip:=searchid((konst));
1599 fap:=lip.idtype; fval:=lip.value;
1600 end
1601 end; (case)
1602 if signed then
1603 if (fap<intptr) and (fap<realptr) and (fap<longptr) then
1604 error(+023)
1605 else if min then fval:=-fval;
1606 (note: negating the v-number for reals and longs)
1607 insyn;
1608 end
1609 else begin fap:=nil; fval:=0 end;
1610 end;
1611
1612 function outinteger(fays:soa; fapisp; err:integer):integer;
1613 var lip:lip; lval_min_max:integer;
1614 begin constant(fays,lip,lval);
1615 if fap<long then
1616 if (errtype=charst) 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 (*****
1627
1628 function typid(arr:integer):sp;
1629 var lip:lip; lval:lip;
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
1637 function simpletp(fays:soa):sp;
1638 var lip:lip; lval:lip; lip_min_max:integer; lval_min_max;
1639 newsubrange:boolean;
1640 begin lip:=nil;
1641 if find((ident..parent),fays,+025) then
1642 if ay<parent then
1643 begin insyn; lip:=top; (decl. const. local to innermost block)
1644 while top<>occur do top:=top.nlink;
1645 lip:=newsp((smallr,wordsize); hip:=nil; max:=0;
1646 repeat lip:=newident((konst,lip,hip,+026);
1647 if lip<nil then
1648 begin enterid(lip);
1649 hip:=lip; lip.value:=max; max:=max+1
1650 until endofloop(fays,(parent),(ident),comma,+027); (+028)
1651 if max<0 then lip.size:=bytesize;
1652 lip.flags:=hip; top:=lip; nextif((parent,+029);
1653 end
1654 else
1655 begin newsubrange:=true;
1656 if ay<ident then
1657 begin lip:=searchid((types,konst)); insyn;
1658 if lip.klasstype then
1659 begin lip:=lip.idtype; newsubrange:=false end
1660 end
1661 begin lip:=lip.idtype; min:=lip.value end
1662 else
1663 constant(fays,(colon2,ident..pluss),lip,min);
1664 if newsubrange then
1665 begin lip:=newsp(subrange,wordsize); lip.subno:=0;
1666 if not minmax(arr) then
1667 begin error(+030); lip:=nil; min:=0 end;
1668 lip.range:=lip;
1669 nextif((colon2,+031); max:=outinteger(fays,lip,+032);
1670 if min>max then begin error(+033); max:=min end;
1671 if (min<0) and (max<0) then lip.size:=bytesize;
1672 lip.min:=min; lip.max:=max
1673 end
1674 end;
1675 simpletp:=lip
1676 end;
1677
1678 function arraytp(fays:soa;
1679 arrtp:structform;
1680 sflag:flagset;

```

```

1681 function element(fays:soa):sp
1682 :sp;
1683 var lip:lip; lval:lip; min_max:integer; ok:boolean; spsy:symbol; lip:lip;
1684 ok:=true;
1685 begin insyn; nextif((break,+034); hsp:=nil;
1686 repeat lip:=newsp(arrtp,0); initpos(lip,arpos);
1687 lip.min:=hsp; hsp:=lip; (link reversed)
1688 if artype=array then
1689 begin spsy:=coloncolon; ok:=ident;
1690 lip:=newident((arrbnd,lip,ail,+035);
1691 if lip<nil then enterid(lip);
1692 nextif((colon2,+036);
1693 lip:=newident((arrbnd,lip,ail,+037);
1694 if lip<nil then enterid(lip);
1695 nextif((colon2,+038); lip:=typid(+039);
1696 ok:=ok and (not (detub(lip)));
1697 end
1698 else
1699 begin spsy:=comma; ok:=((ident..parent);
1700 lip:=simpletp(fays,(comma,break,ofy,ident..packedy));
1701 ok:=bounds(lip,min_max)
1702 end;
1703 if not ok then begin error(+040); lip:=nil end;
1704 lip.lntype:=lip
1705 until endofloop(fays,(break,ofy,ident..packedy),ok,
1706 spsy,+041); (+042)
1707 nextif((break,+043); nextif((ofy,+044);
1708 lip:=element(fays);
1709 if lip<nil then sflag:=sflag + lip.sflag + [withfile];
1710 repeat (reverse links and compute aim)
1711 lip:=lip.nlink; hsp:=lip; hsp.sflag:=sflag;
1712 if artype=array then hsp.size:=arraysize(hsp,sflag in sflag);
1713 lip:=hsp; hsp:=lip;
1714 until hsp=nil; (lip points to array with highest dimension)
1715 arraytp:=lip
1716 end;
1717
1718 function typ(fays:soa):sp;
1719 var lip:lip; lval:lip; min_max:integer;
1720 sflag:flagset; lip:lip;
1721
1722 function fldlist(fays:soa):sp;
1723 (level 2: << typ)
1724 var lip:lip; lval:lip; lip:lip;
1725
1726 function varpart(fays:soa):sp;
1727 (level 3: << fldlist << typ)
1728 var lip:lip; lval:lip; lip:lip; hsp:=lip; vsp:=lip; top:=lip;
1729 min_max:=int; nvar:=integer; lid:=alpha;
1730 begin insyn; lip:=nil; lip:=nil;
1731 lip:=newsp(hsp,0);
1732 if ay<ident then error(+045) else
1733 begin lid:=id; insyn;
1734 if ay<colon then
1735 begin lip:=newsp(field,lid,ail,ail); enterid(lip); insyn;
1736 if ay<ident then error(+046) else

```

```

1737 begin lid:=id; insyn end;
1738 end;
1739 if ay<ofy then (otherwise you may destroy id)
1740 begin lid:=lid; lip:=searchid((types)) end;
1741 end;
1742 if lip<nil then tfap:=nil else tfap:=lip.idtype;
1743 if bounds(tfap,int,nvar) then nvar:=nvar-int+1 else
1744 begin nvar:=0;
1745 if tfap<nil then begin error(+047); tfap:=nil end
1746 end;
1747 lip:=lip; tfap:=tfap;
1748 if tip<nil then (explicit tag)
1749 begin tip:=tfap;
1750 tip.flags:=address(of,sizeof(tfap),sflag in sflag)
1751 end;
1752 nextif((ofy,+048); alnoc:=oc; nvar:=nvar; headsp:=nil;
1753 repeat hsp:=nil; (for each caselabel list)
1754 repeat nvar:=nvar-1;
1755 int:=outinteger(fays,(ident..pluss,comma,colon1,parent,
1756 semicolon,comma,rparent),tfap,+049);
1757 lip:=headsp; (each label may occur only once)
1758 while lip<nil do
1759 begin if lip.varval=nt then error(+050);
1760 lip:=lip.nvar
1761 end;
1762 vsp:=newsp(variant,0); vsp.varval:=int;
1763 vsp.nvar:=headsp; headsp:=vsp; (chain of case labels)
1764 vsp.subst:=hsp; hsp:=vsp;
1765 (use this field to link labels with same variant)
1766 until endofloop(fays,(colon1,parent,semicolon,comma,rparent),
1767 (ident..pluss),comma,+051); (+052)
1768 nextif((colon1,+053); nextif((parent,+054);
1769 top:=fldlist(fays,(parent,semicolon,ident..pluss));
1770 if oc=nvar then nvar:=oc;
1771 while vsp<nil do
1772 begin vsp:=nil; oc:=hsp:=vsp.subst;
1773 vsp:=vsp.subst:=top; top:=hsp
1774 end;
1775 nextif((parent,+055);
1776 oc:=minoc;
1777 until lastsemicolon(fays,(ident..pluss),+056); (+057 +058)
1778 if nvar>0 then error(+059);
1779 top.flags:=headsp; top.size:=minoc; oc:=minoc; vsp:=top;
1780 end;
1781
1782 begin (fldlist)
1783 if find((ident),fays,(array),+060) then
1784 repeat lip:=nil; hip:=nil;
1785 repeat lip:=newident((field,ail,ail,+061);
1786 if lip<nil then
1787 begin enterid(lip);
1788 if lip<nil then hip:=lip else lip:=lip.next:=lip; lip:=lip;
1789 end;
1790 until endofloop(fays,(colon1,ident..packedy,semicolon,comma),
1791 (ident),comma,+062); (+063)
1792 nextif((colon1,+064);

```

```

1793   lsp:=typ(fays[caseary,semicolon]);
1794   if lsp<0 then if withfile in lsp then error(-(1078));
1795   sflag:=sflag<withfile;
1796   while hip<0 do
1797     begin hip:=lsp;
1798     hip:=f.offset+address(0, sizeof(lsp), speak in sflag);
1799     hip:=hip.next
1800     end;
1801   until lastsemicolon(fays[caseary],[ident],*065); [*066 *067]
1802   if sycaseary then fidlist:=vpart(fays) else fidlist:=nil;
1803 end;

```

```

1806 begin [typ]
1807   sflag:=[]; lsp:=nil;
1808   if sycaseary then begin sflag:=[speak]; inays end;
1809   if find1((ident..filey),fays,*068) then
1810     if sy in [ident..arrow] then
1811       begin if speak in sflag then error(*069);
1812         if sycaseary then
1813           begin lsp:=newip(pointer,ptrsize); inays;
1814           if not intypedec then lsp:=lsp+stypid(*070) else
1815             if sy<0 then error(*071) else
1816               begin f.pptr:=newip(types,id,lsp,f.pptr); inays end
1817             end
1818           else lsp:=slmptyp(fays);
1819           end
1820         else
1821           case sy of
1822             <<<<<<<<<<<<<
1823           arrayy:
1824             lsp:=arrstyp(fays,arrayy,sflag,typ);
1825           recordy:
1826             begin inays;
1827               new(lsp,rec); lsp:=rec; lsp:=lsp.next;
1828               lsp:=f.name; lsp:=lsp;
1829               on:=0; lsp:=fidlist(fays[anday]); [fidlist updates on]
1830               lsp:=newip(record,0); lsp:=lsp;
1831               lsp:=f.pptr; lsp:=lsp; lsp:=lsp;
1832               top:=top; link:=nextif(endsay,*072)
1833             end;
1834           sety:
1835             begin inays; nextif(endsay,*073); lsp:=slmptyp(fays);
1836             if bounds(lsp,min,max) then lsp:=newip(lsp) else
1837               if lsp:=lsp then
1838                 begin error(*074); max:=lsp-1 end
1839               else
1840                 begin error(*075); lsp:=nil end;
1841               if lsp:=lsp then
1842                 begin if lsp:=lsp then
1843                   begin if bounds(lsp,min,max) then [nothing];
1844                   s:=s+max;
1845                   if (min<0) or (max>sz) or (sz div by 8) = 0 then error(*076) else
1846                     lsp:=newip(power,sz div by 8); lsp:=lsp; s:=s;
1847                   lsp:=newip(power,sz div by 8); lsp:=lsp;
1848                 end;
1849               filey:
1850                 begin inays; nextif(endsay,*077); lsp:=styp(fays);

```

```

1849   if lsp<0 then if withfile in lsp then error(-(1078));
1850   sz:=sizeof(lsp); if sz>buffsize then sz:=buffsize;
1851   lsp:=newip(filey,sz,ftype); lsp:=lsp;
1852   end;
1853   (>>>>>>>>>>>>)
1854   end; (case)
1855   typ:=lsp;
1856 end;

```

```

1858 function vpartyp(fays:soa):sp;
1859 begin
1860   if find2([arrayy],fays=[ident],*079) then
1861     vpartyp:=arrstyp(fays, arrayy,[],vpartyp)
1862   else
1863     vpartyp:=stypid(*080)
1864 end;

```

```

1866 [=====]

```

```

1868 procedure block(fays:soa; fip:ip); forward;
1869 [pdeclaration calls block. With a more obscure lexical
1870 structure this forward declaration can be avoided]

```

```

1872 procedure labeldeclaration(fays:soa);
1873 var lip:ip;
1874 begin with b do begin
1875   repeat
1876     if sy<0 then error(*081) else
1877       begin
1878         if searchlab(lchain,val) then error(*082, val) else
1879           begin new(lip); lip:=labval;
1880             if val>999 then teststandard;
1881             libno:=libno+1; lip:=labname+libno; lip:=lip;
1882             lip:=seen:=false; lip:=nextlab; lchain:=lip;
1883             end;
1884             inays
1885           end
1886         until endofloop(fays=[semicolon],[intest],comma,*083); [*084]
1887         nextif(semicolon,*085)
1888       end end;

```

```

1890 procedure constdefinition(fays:soa);
1891 var lip:ip;
1892 begin
1893   repeat lip:=newident(const,nil,nil,*086);
1894     if lip<0 then
1895       begin nextif(eqy,*087);
1896         constant(fays=[semicolon],[ident],lip:=lsp:=lsp);
1897         nextif(semicolon,*088); enterid(lip);
1898         end;
1899     until not find2((ident),fays,*089);
1900   end;

```

```

1902 procedure typedefinition(fays:soa);
1903 var lip:ip;
1904 begin f.pptr:=nil; intypedec:=true;

```

```

1905   repeat lip:=newident(types,nil,nil,*090);
1906     if lip<0 then
1907       begin nextif(eqy,*091);
1908         lip:=lsp:=styp(fays=[semicolon],[ident]);
1909         nextif(semicolon,*092); enterid(lip);
1910         end;
1911     until not find2((ident),fays,*093);
1912     while f.pptr<0 do
1913       begin assert sy<0;
1914         id:=f.pptr:=newip(lsp,searchid(ftypes));
1915         f.pptr:=id; ltype:=lip:=lsp; f.pptr:=f.pptr.next
1916         end;
1917     intypedec:=false;
1918   end;

```

```

1920 procedure vardclaration(fays:soa);
1921 var lip,hip,vip:ip; lsp:=lsp;
1922 begin with b do begin
1923   repeat lip:=nil; vip:=nil;
1924     repeat vip:=newident(vars,nil,nil,*094);
1925       if vip<0 then
1926         begin enterid(vip); vip:=lsp;
1927           if lip:=lsp then lip:=vip else lip:=vip;
1928         end;
1929     until endofloop(fays=[colon],[ident],packeday,[ident],comma,*095);
1930     [*096]
1931     nextif(colon,*097);
1932     lsp:=styp(fays=[semicolon],[ident]);
1933     while hip<0 do
1934       begin hip:=lsp;
1935         hip:=hip:=address(0,sizeof(lsp),false); hip:=hip.next
1936         end;
1937     nextif(semicolon,*098);
1938     until not find2((ident),fays,*099);
1939   end end;

```

```

1941 procedure pthead(fays:soa);
1942 var lip:ip;
1943 var again:boolean;
1944 param:boolean; forward;

```

```

1946 function parlist(fays:soa; var hlist:integer):ip;
1947 var lastip,hip,lip,ptp:ip; lsp:=lsp; fflag:=lsp:=lsp; again:boolean;
1948 sz:=integer;
1949 begin parlist:=nil; lastip:=nil;
1950 repeat (case for each formal-parameter-section)
1951   if find1((ident..var..proc..func..sym),fays=[semicolon],*0100) then
1952     begin
1953       if (sycaseary) or (sycfunc) then
1954         begin
1955           pthead(fays=[semicolon],[ident..var..proc..func..sym],
1956             hip:=lsp:=lsp;
1957             hip:=p.pos:=address(hlc,param:=ptrsize,false);
1958             hip:=p:=lsp; ltype:=lsp;
1959             top:=top; link:=level-1
1960           end

```

```

1961   else
1962     begin hip:=nil; lip:=nil; fflag:=assigned(none);
1963     if sycaseary then
1964       begin fflag:=refer_assigned_used(none); inays end;
1965     repeat lip:=newident(vars,nil,nil,*0101);
1966       if lip<0 then
1967         begin enterid(lip); lip:=lsp;
1968           if lip:=lsp then lip:=vip else lip:=vip;
1969           lip:=lsp;
1970           end;
1971         if fflag=[assigned] then
1972           until endofloop(fays=[semicolon],[ident],comma,*0102); [*0103]
1973         nextif(colon,*0104);
1974         if refer in fflag then
1975           begin lip:=vpartyp(fays=[semicolon]);
1976             s:=ptrsize; lip:=lsp;
1977             while formof(lsp,[array]) do
1978               begin lip:=newip(address(hlc,sz,false);
1979                 lip:=lsp;
1980               end;
1981             end
1982           else
1983             begin lip:=stypid(*0105); sz:=sizeof(lsp);
1984               lip:=lip;
1985               while lip<0 do
1986                 begin lip:=newip(address(hlc,sz,false);
1987                   lip:=lip;
1988                 end;
1989               end;
1990           if lastip then parlist:=lip else lastip:=lip;
1991           lip:=lsp;
1992           end;
1993         until endofloop(fays=[ident..var..proc..func..sym],
1994           semicolon,*0106); [*0107]
1995       end;
1996     end;

```

```

1998 procedure pthead; [forward declared]
1999 var lip:ip; lsp:ip; ltype:ip; kl:integer;
2000 begin lip:=nil; again:=false;
2001 if sycproc then kl:=p:=lsp else
2002   kl:=f:=lsp; fays:=fays=[colon],[ident] end;
2003 inays;
2004 if sy<0 then begin error(*0108); id:=sz:=0 end;
2005 if not param then lip:=searchsection(top,fname);
2006 if lip<0 then
2007   if (lip:=lsp) or (lip:=lsp) then
2008     error(*0109, id)
2009   else
2010     begin b:=formof(lsp,[array]); again:=true end;
2011   if again then inays else
2012     begin lip:=newip(kl,id,nil,nil);
2013     if sy<0 then begin error(*0110);
2014       lastip:=lsp; ltype:=lsp;
2015     end;
2016     level:=level+1;

```

