

*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-1AS 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 V. 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.V.Stevensen & H. van Steyvenan,
48 "Description of a 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 boolsize = bytesize;
119 charsize = bytesize;
120 intsize = shortsize;
121 buffsize = 512;
122 maxsetsize = 4096; [t15 div bytesize]
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 misetlnt = 0;
141 maxsetint = 15; {default}
142
143 {constants describing the compact EM1 code}
144 MAGICLOW = 172;
145 MAGICHIGH = 0;
146 memerror = 0;
147 memoptimal = 1;
148 memoptimal = 2;
149 memreg = 3;
150 memlino = 4;
151 memfloats = 5;
152
153 {ASCII characters}
154 Lab = 9;
155 newline = 10;
156 horlab = 13;
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 spocess = ;
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 t4 = 256;
78 t8 = 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 bytesize = 1;
91 wordsize = 2;
92 addrsize = wordsize;
93 punysize = wordsize;
94 shortsize = wordsize;
95 longsize = 4;
96
97 #ifdef SFLDPT
98 floatsize = 4;
99 #endif
100 #ifdef SFLOAT
101 floatsize = 8;
102 #endif
103
104 {Pascal sizes. For ptrsize, realsize and floatsize see handleopt}
105 { EM-1 requires that objects greater than a single byte start at a
106 word boundary, so their address is even. Normally, a full word
107 is also allocated for objects of a single byte. This extra byte
108 is really allocated to the object, not only striped by alignment,
109 i.e. if the value false is assigned to a boolean variable then
110 both bytes are cleared. For single byte objects in packed arrays
111 or packed records, however, only one byte is allocated, even if
112 the next byte is unused. Strings are packed arrays. The size of
113 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 intoct,charoct,realoct,longoct,stringoct,alioct,minisy,
173 plusy,percent,arrow,arrayy,recordy,setsy,filesy,
174 packedy,progy,labely,consty,typesy,varsy,procsy,
175 funcy,begisy,gotosy,ifsy,whilesy,repeaty,foray,
176 withay,casay,becomes,staray,divy,mody,alsay,
177 anday,oray,exay,notay,asy,asy,asy,
178 laby,laby,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,odotoch,periodch,lesoch,
183 greaterch,lparentch,lbracoch,
184 ); {different entries}
185 rparentch,lbrackch,rbrackch,comoch,semoch,arrowch,
186 plusch,minch,alch,star,equal,
187 ); {also symbols}
188 others
189 );
190
191 standpf = (pread,preadln,write,partials,pput,pgot,
192 preat,prewrite,pnes,pdispose,ppack,punpack,
193 pmack,prelease,ppage,phalt,
194 ); {all procedures}
195 foof,fooln,foab,foqr,ford,fofr,fpred,foact,foadd,
196 furmo,fround,fsin,foos,foexp,foqtr,fit,foctan
197 ); {all functions}
198 ); {the order is important}
199
200 libnames = (INL,IFL,CLS,VM,
201 OFN,GETX,END,IBC,END,ML,ML,
202 ); {see input files}
203
204 CRF,FUT,URI,MSI,MAC,MSC,MRS,MSS,MRB,
205 MMB,MRB,MRM,MRM,MSL,MSP,MR2,MS2,MLM,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,BTS,HEX,S,SAV,EST,IDI,INT,
212 ASS,OTO,PAC,UMP,DLS,ARZ,MDI,MDL,
213 ); {miscellaneous}
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
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
```

PASCAL NEWS #22 & #23 SEPTEMBER, 1981 PAGE 4 PASCAL NEWS #22 & #23 SEPTEMBER, 1981 PAGE 5

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 lpb= "label; 286 (first name: root of tree)

232 bpb= "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 lcin:integer; (data location counter (from begin of proc))

295 libno: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 : (elast:sp; (type of set elements)

313 files : (fitype:sp; (type of file elements)

314 arrays:array; (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:linkip; (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:stndp); (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 label: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 lablib: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 sstr: (type,access method,position,value of expr)

379 (returned by inqpr)

380 chchar: (last character)

381 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 idalpha: (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 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,inptr,txtptr,emptset,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 progrid: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 bblockinfo: (all info to be stored at pfdeclaration)

421 error: (all info required for error messages)

422 fstr: (fstr 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 rsv:array(r:range) of symbol;

433 (symbol for reserved words)

434 carray[char] of chartype;

435 (char type of a character)

436 carray[pr:peratch, equal] of symbol;

437 (symbol for single character symbols)

438 lms:array[libnum] of array[1..4] of char;

439 (mnemonics of pascal library routines)

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

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

442 (for different options)

443 undefip:array[ideclass] of ip;

444 (used in searchid)

445 arv:array(0..maxarg) of

446 record name:alpha; ad:integer end;

447 (here here the external heading names)

448 (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,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:= ' '; 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_lin,e.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:=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_mes:=spaces; e_mes[1:]=1; puterr(err) end;
542  procedure errid(err:integer; var id:alpha);
543  begin e_mes:=id; e_mes[1:]=1; puterr(err) end;
545  procedure errint(err:integer; i:integer);
546  begin e_mes:=i; e_mes:=spaces; puterr(err) end;
548  procedure aspperr(err:integer);
549  begin if e_spp<all then begin error(err); e_spp:=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:idelase; 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_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.p_llink:=nil; p.p_rlink:=nil; p.p_iflag:=f; newp:=p
611  end;
613  function newp(sf:structform; sz:integer):sp;
614  var p:sp; 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 new(p,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 c:char;
641  begin
642  (initialize the first name space)
643  new(top,blk); top.p_occu:=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  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))::=staboh;
685   os(' ')::=layout;   os('!')::=dqotech;   os('\"')::=qotech;
686   os('@')::=parantoh; os('#')::=rparantoh; os('$')::=star;
687   os('%')::=splunch;  os('&')::=soomach;   os('&')::=minoh;
688   os('\"')::=paridoch; os('\'')::=slash;   os('~')::=wooloh;
689   os('`')::=smich;    os('<')::=lessoch;  os('=')::=equal;
690   os('~')::=grateroh; os('[')::=lbrackoh; os(']')::=rbrackoh;
691   os('^')::=arrouh;   os('`')::=lbrackoh; os('`')::=rbrackoh;
692   (single character symbols in char type order)
693   os('p')::=parantoh; os('r')::=rparantoh; os('s')::=sbrackoh;
694   os('b')::=lbrackoh; os('B')::=rbrackoh; os('c')::=comma;
695   os('o')::=omoh;     os('O')::=oomah;   os('n')::=narrow;
696   os('l')::=lshoh;   os('L')::=lshay;   os('m')::=mshay;
697   os('a')::=ashoh;   os('A')::=ashay;   os('e')::=equal;
698   os('e')::=eqsh;   os('E')::=eqsh;
699   end;

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   underfip[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^.foont:=p;
727   p:=newip(const,'maxint','intptr,nil); p^.value:=maxint; enterid(p);
728   p:=newip(const,'spaces,chars,nil); p^.value:=maxcharord;

```

```

729   charptr^.foont:=p;
730   end;

731   procedure init3;
732   var j:standpf; p:ip; q:mp;
733   p:=array(standpf) of alpha;
734   ftype:=array(foef..farotan) of sp;
735   begin
736   (names of standard procedures/functions)
737   p[read] :='read'; p[readin] :='readin';
738   p[write] :='write'; p[writein] :='writein';
739   p[put] :='put'; p[putin] :='putin';
740   p[page] :='page'; p[pgot] :='got';
741   p[rewrite] :='rewrite'; p[reset] :='reset';
742   p[dispose] :='dispose'; p[pack] :='pack';
743   p[unpack] :='unpack'; p[mark] :='mark';
744   p[release] :='release'; p[halt] :='halt';
745   p[ord] :='ord'; p[foin] :='foin';
746   p[feor] :='feor'; p[foab] :='foab';
747   p[foord] :='foord'; p[fofgr] :='fofgr';
748   p[fofprad] :='fofprad'; p[fofuc] :='fofuc';
749   p[fofod] :='fofod'; p[fofrou] :='fofrou';
750   p[fofrou] :='fofrou'; p[fofou] :='fofou';
751   p[fofgrt] :='fofgrt'; p[fofexp] :='fofexp';
752   p[fofarc] :='fofarc'; p[fofarc] :='fofarc';
753   (parameter types of standard functions)
754   ftype[foef] :=nil; ftype[foein] :=nil;
755   ftype[foab] :=nil; ftype[fofgr] :=nil;
756   ftype[fofprad] :=nil; ftype[fofuc] :=intptr;
757   ftype[fofod] :=intptr; ftype[fofrou] :=nil;
758   ftype[fofou] :=nil; ftype[fofexp] :=realptr;
759   ftype[fofarc] :=realptr; ftype[fofarc] :=realptr;
760   (standard procedure/function identifiers)
761   for j:=read to halt do
762   begin new(p,proc,standpf); p^.klass:=proc;
763   p^.name:=p[j]; p^.pkind:=standpf; p^.key:=j; enterid(p);
764   end;
765   for j:=foef to farotan do
766   begin new(p,proc,standpf); p^.klass:=fno; p^.idtype:=ftype[j];
767   (idtype is used not for result type but for parameter type if)
768   p^.name:=p[j]; p^.pkind:=standpf; p^.key:=j; enterid(p);
769   end;
770   (program identifier)
771   prog:=newip(proc,'main','nil,nil);
772   (new name space for user external)
773   new(blck); q:=occur:blck; q^.allink:=top; q^.fname:=nil; top:=q;
774   end;

781   procedure init4;
782   var c:char;
783   begin
784   (pascal library monom(c))

```

```

785   lnn[EL] :='el'; lnn[EFL] :='efl'; lnn[CLS] :='cls';
786   lnn[VM] :='vm'; lnn[GT] :='gt'; lnn[ROI] :='roi';
787   lnn[OP] :='op'; lnn[GRX] :='grx'; lnn[RDI] :='rdi';
788   lnn[RDC] :='rdc'; lnn[RDE] :='rde'; lnn[RDL] :='rdl';
789   lnn[RL] :='rl'; lnn[PUI] :='pui'; lnn[WR] :='wr';
790   lnn[CR] :='cr'; lnn[WC] :='wc'; lnn[WS] :='ws';
791   lnn[WI] :='wi'; lnn[WR] :='wr'; lnn[WS] :='ws';
792   lnn[VBS] :='vbs'; lnn[VSS] :='vss'; lnn[VWB] :='vwb';
793   lnn[VSB] :='vsb'; lnn[VSR] :='vsr'; lnn[VSN] :='vsn';
794   lnn[WL] :='wl'; lnn[WS] :='ws'; lnn[WS] :='ws';
795   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
796   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
797   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
798   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
799   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
800   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
801   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
802   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
803   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
804   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
805   lnn[WR] :='wr'; lnn[WR] :='wr'; lnn[WR] :='wr';
806   (options)
807   for c:=a to z do begin opt[c]:=0; forceopt[c]:=false end;
808   opt['a']:=0; (floats div wordsize; (default real size in words)
809   opt['i']:=maxint+1;
810   opt['l']:=0;
811   opt['r']:=0;
812   opt['p']:=0; (pointers div wordsize; (default pointer size in words)
813   opt['s']:=0;
814   opt['t']:=0;
815   opt:=off;
816   (scalar variables)
817   b:=nil;
818   b.l:=0;
819   b.l:=0;
820   b.l:=0;
821   b.l:=0;
822   e:=nil;
823   e.l:=0;
824   e.l:=0;
825   e.l:=0;
826   e.l:=0;
827   e.l:=0;
828   l:=0;
829   d:=0;
830   s:=0;
831   l:=0;
832   s:=0;
833   l:=0;
834   s:=0;
835   l:=0;
836   s:=0;
837   l:=0;
838   s:=0;
839   l:=0;
840   s:=0;
841   l:=0;
842   s:=0;

```

```

843   argv[0].ed:=1;
844   end;

845   procedure handleopts;
846   begin
847   opt:=opt['a'];
848   dopt:=opt['d'];
849   lopt:=opt['l'];
850   sopt:=opt['s'];
851   realize:=opt['r'] * wordsize; realptr^.size:=realize;
852   ptrsize:=opt['p'] * wordsize; nilptr^.size:=ptrsize;
853   fsize:=dintsize + 2*ptrsize;
854   textptr^.size:=fsize+bufsize; stringptr^.size:=ptrsize;
855   if sopt<off then begin dopt:=off; dopt:=off end;
856   else if opt['u']<off then os(' ');
857   if dopt<off then enterid(newip(types,'string','stringptr,nil));
858   if dopt<off then enterid(newip(types,'long','longptr,nil));
859   if opt['o']<off then begin gen(p,mes,mesoptoff); genend end;
860   if ptrsize<wordsize then begin gen(p,mes,mesvirtual); genend end;
861   if dopt<off then ftype:=true; (temporary kludge)
862   end;

863   (=====)
864   procedure trace(tname:alpha; fip:ip; var nandib:integer);
865   var i:integer;
866   begin
867   if opt['t']<off then
868   begin
869   if nandib=0 then
870   begin dibo:=dibo+1; nandib:=dibo; genld(dibo);
871   gen0(p,mes); write(m,sp,ocn,8);
872   for i:=1 to 8 do write(m,ord(fip^.name[i])); genend;
873   end;
874   gen1(sp,ort,0); gen0(sp,mes,nandib);
875   gen0(sp,ocn); genid(m,sp,mes,tname);
876   end;
877   end;

880   function formof(fsp:sp; form:formset):boolean;
881   begin if fsp=nil then formof:=false else formof:=fsp^.form in forms end;

883   function sincf(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   sincf:=s;
889   end;

891   function even(i:integer):integer;
892   begin if odd(i) then i:=i+1; even:=i end;

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;
901   procedure setop(s:byte);
902   begin gen!(s,even(sizeof(s.asp))) end;
904   procedure spendemptyst(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_lae,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 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;
947   procedure deaddr(var p:position);
948   begin if p.lv=0 then gen!(op_lae,p.ad) else loadpos(p,ptrsize) end;
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);
959                 #endif
960               gen!(op_lae,ad)
961             else
962               gen!(op_lae,ad)
963             else
964               if lv=level then gen!(op_lal,ad) else
965                 begin gen!(op_lex,level-lv); gen!(op_mdi,ad) end;
966             loadpos(pos,ptrsize);
967             ploaded:=
968             ;
969             indexed:
970               gen!(op_sas);
971             end; [case]
972             sk:=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 of
981         cat:
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         sk:=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 of
1003         fixed:
1004           with pos do
1005             if lv<0 then
1006               #ifdef SECTIONS
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     begin 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 (sk=fixed) and not packbit then pos.ad:=pos.ad+off else
1029       begin loadadr; gen!(op_mdi,off) end
1030   end;
1032   procedure loadheap;
1033   begin if forsof(s.asp,[arrays..records]) then loadadr else load end;
1035   {.....}
1037   procedure nextch;
1038   begin
1039     col:=col+1; read(input,cb); e.col:=e.col+1; chay:=col[ch];
1040   end;
1042   procedure nextln;
1043   begin
1044     if eof(input) then
1045       begin
1046         if not eofexpected then error(+03) else
1047           begin
1048             if flused 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.col:=0; e.ln:=e.ln+1; e.lnr:=e.lnr+1;
1059   if not including then
1060     begin e.orig:=orig; gval:=struc end;
1061   end;
1063   procedure options(normal:boolean);
1064   var o:stchar; i:integer;
1066   procedure goto;
1067   var b:byte;
1068   begin
1069     if normal then
1070       begin nextch; o:=ch end
1071     else
1072       begin read(am,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 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 nextch until (ch=' ') or eof;
1098     while ch<digit do
1099       begin i:=i*10 + ord(ch) - ord('0'); nextch end;
1100     while (ch=' ') and not eof do nextch;
1101     if (ch='*') or (eof) then error(+04) else
1102       begin nextch;
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             nextch
1110           end;
1111       if source=emptyfnam then source:=e.fnam;
1112       including:=source<='.';
1113       i:=i-1; e.lnr:=i;
1114       if not including then e.orig:=i
1115     end;
1116     while not eof do nextch;
1117   end;
1119   procedure putdig;
1120   begin i:=i+1; if i<max then strbuff[i]:=ch; nextch end;

```



```

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.iff; asp:=realptr; fltused:=true end;
1369   ri: begin op:=op.aff; asp:=intptr; fltused:=true end;
1370   li: begin op:=op.oid; asp:=longptr end;
1371   ll: begin op:=op.odd; asp:=intptr end;
1372   lr: begin op:=op.odd; asp:=realptr; fltused:=true end;
1373   rl: begin op:=op.odd; asp:=longptr; fltused:=true end;
1374   end;
1375   gen0(op)
1376 end end;
1377
1378 procedure negate(l1:integer);
1379 var l2:integer;
1380 begin
1381   if a.asp=ptr 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=realptr) 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:=ll else
1468       if (p=longptr) and (q=intptr) then compat:=ll 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.aset,q.aset) <= 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   ll:
1497     begin oponent(ts); checkasp(fsp,err) end;
1498   lr,rl,lr,lr:
1499     oponent(ts);
1500   set:
1499     asppemptyset(fsp);
1501   noteq,ri,se:
1502     aspperr(err);
1503   end;
1504 end;
1505
1506 procedure force(fsp:sp; err:integer);
1507 begin load; checkasp(fsp,err) end;
1508
1509
1510 function neident(kl:ld;id:sp; ntp:nt; err:integer):lp;
1511 begin neident:=null;
1512 if sp<=ident then error(err) else

```

```

1401 function bounds(fsp:sp; var fln,fmx:integer):boolean;
1402 (compute bounds if possible, else return false)
1403 begin bounds:=false; fln:=0; fmx:=0;
1404 if fsp<=all then
1405   if fsp.for= subrange then
1406     begin fln:=fsp.min; fmx:=fsp.max; bounds:=true end else
1407   if fsp.for=scalar then
1408     if fsp.foomat<=0 then
1409       begin fln:=0; fmx:=fsp.foomat.value; bounds:=true end
1410   end;
1411
1412 procedure genrok(fsp:sp);
1413 var min,max,ano:integer;
1414 begin
1415   if opt['r']<=off then if bounds(fsp,min,max) then
1416     begin
1417       if fsp.for=scalar then ano:=fsp.scalno else ano:=fsp.subrno;
1418       if ano=0 then
1419         begin dbno:=dbno+1; ano:=dbno;
1420         genib(dbno); genl(pe_rom,min); genot(max); genod;
1421         if fsp.for=scalar then fsp.scalno:=ano else
1422         fsp.subrno:=ano
1423         end;
1424       end;
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,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 l=sp.for=scalar then
1451         if l=sp.rangetype=ptr then
1452           if l=sp.min=1 then
1453             string:=true
1454         end;
1455 end;

```

```

1513   begin neident:=newip(kl.id,nt,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:=speak;
1521   l:=asetype.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:=asetype(fsp,asetype);
1543 if not pack then sz:=even(sz);
1544 if bounds(fsp.inctype,min,max) then (we checked before)
1545   dbno:=dbno+1; fsp.arpos.lv:=0; fsp.arpos.ed:=dbno;
1546   genib(dbno); 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.klass=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,asetype(lsp,ord(formof(lsp,pointer))));
1564       if flp<=all then if withfile in l.sflag then
1565         if l=sp.files then
1566           if level=1 then
1567             begin
1568               for i:=2 to argo do with argo[i] do

```





