

*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<br>(for PN 15..18)                     |                | G. Marshall             |
| 4  | APPLICATIONS  |                |                         |
| 4  | The FMI Compiler (code)   |                | A. Tanenbaum            |
| 38 | Options -- Control Statement<br>Option Settings                   |                | S. Leonard              |
| 39 | Treeprint -- Prints Trees<br>on a Character Printer               |                | Freed & Carosso         |
| 44 | Compress & Recall -- Text<br>compression using Huffman codes      |                | T. Stone                |
| 50 | ARTICLES  |                |                         |
| 50 | "The Performance of three CP/M based<br>Translators"              |                | Johnson &<br>Sidebottom |
| 54 | "A Geographer Teaches Pascal --<br>Reflections on the Experience" |                | J. Pitzl                |
| 56 | "An Extension That Solves Four<br>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 |



225 rrange= 0..rdim;  
 226 bytes 0..lbit;

228 (pointer types)  
 229 sp= "structure;  
 230 ip= "identifier;  
 231 lpa= "label;  
 232 bpa= "blockinfo;  
 233 npa= "nameinfo;

235 (set types)  
 236 set= set of symbol;  
 237 setofid= set of idclass;  
 238 format= set of structform;  
 239 sflagset= set of structflag;  
 240 iflagset= set of idclass;

242 (array types)  
 243 alpha= packed array[irange] of char;  
 244 ftype= packed array[frange] of char;

246 (record types)  
 247 rrec= record  
 248 rno: integer; {array number}  
 249 rsi: integer; {identifier parameter if required}  
 250 rmi: integer; {numeric parameter if required}  
 251 rco: integer; {column number}  
 252 rli: integer; {line number}  
 253 rri: integer; {relative to start of (included) file}  
 254 rfi: integer; {file, but before preprocessing}  
 255 rsn: string; {source file name}  
 256 end;

258 position= record  
 259 ad: integer; {the addr info of certain variable}  
 260 lv: integer; {the level of the beast}  
 261 #ifdef SEGMENTS  
 262 sgrange= {only relevant for globals (lv=0)}  
 263 #endif  
 264 end;

266 (records of type attr are used to remember qualities of  
 267 expression parts to delay the loading of them.  
 268 Reasons to delay the loading of one word constants:  
 269 - bound checking  
 270 - set building.  
 271 Reasons to delay the loading of direct accessible objects:  
 272 - efficient handling of read/write  
 273 - efficient handling of the with statement.  
 274  
 275 )  
 276 attr= record  
 277 exp: expression; {type of expression}  
 278 packbit: boolean; {true for packed elements}  
 279 attrkind: access method; {access method}  
 280 pos: position; {eg. lv and ad}  
 281 (if almost then the value is stored in ad)

281 end;

283 nameinfo= record {one for each separate name space}  
 284 nlink: ip; {one deeper}  
 285 fname: ip; {first name: root of tree}

286 case occur= where of  
 287 blk: ();  
 288 rec: ();  
 289 area: (w: string) {name space opened by with statement}  
 290 end;

292 blockinfo= record {all info of the current procedure}  
 293 blkpp: ip; {pointer to blockinfo of surrounding proc}  
 294 lc: integer; {data location counter (from begin of proc)}  
 295 lbno: integer; {number of last local label}  
 296 forcount: 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: (scain: 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 sub descriptor}  
 310 );  
 311 pointer: (atype: ip; {type of pointed object}  
 312 power: (elast: ip; {type of set elements}  
 313 files: (fitype: ip; {type of file elements}  
 314 );  
 315 arrays: orarray; {type of array elements}  
 316 );  
 317 intype: ip; {type of array index}  
 318 arpos: position; {position of array descriptor}  
 319 );  
 320 records: (fstrid: ip; {points to first field}  
 321 tagap: ip; {points to tag if present}  
 322 );  
 323 variant: (varval: integer; {tag value for this variant}  
 324 mtrvar: ip; {next equilevel variant}  
 325 subtag: ip; {points to tag for sub-case}  
 326 );  
 327 tag: (fstrvar: ip; {first variant of case}  
 328 tfrid: ip; {type of tag}  
 329 );  
 330 end;

331 identifier= record  
 332 idtype: ip; {type of identifier}  
 333 name: alpha; {name of identifier}  
 334 link: rlink: ip; {see enterid, searchid}  
 335 next: ip; {used to make several chains}  
 336 iflag: iflagset; {several flag bits}

337 case class: idclass 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 pfkind: kind of pf of  
 351 standard: (key: standard pf); {identification}  
 352 formal, actual, forward, extra: (lv gives declaration level.  
 353 pfpos: position); {lv gives instruction segment of this proc and  
 354 ad 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; {ls when heading summed}  
 362 )  
 363 )  
 364 end;

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 labdir: integer; {true means only locally used,  
 373 otherwise dlbno of label information}  
 374 end;

376 (-----)  
 377 var {the most frequent used externals are declared first}  
 378 sy: symbol; {last symbol}  
 379 sstr: (type, access method, position, value of expr)  
 380 returned by inexpr)  
 381 ch: character; {last character}  
 382 chtype: (type of ch, used by inexpr)  
 383 val: integer; {if last symbol is an constant }  
 384 len: integer; {string length}  
 385 col: boolean; {true if current ch replaces a newline}  
 386 newstr: boolean; {true for strings in " "  
 387 id: alpha; {if last symbol is an identifier}  
 388 (some counters)  
 389 line: integer; {line number on code file (1..n)}  
 390 dlbno: integer; {number of last global number}  
 391 lmax: integer; {deepest level of nesting of ls}  
 392 level: integer; {current static level}

393 ptrsize: integer;  
 394 realize: integer; {file header size}  
 395 rsize: integer; {index in args}  
 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 sopt: integer; {standard option}  
 401 (pointers pointing to standard types)  
 402 realptr, intptr, textptr, emptyptr, boolptr: ip;  
 403 charptr, nilptr, stringptr, longptr: ip;  
 404 )  
 405 (flags)  
 406 giveline: boolean; {give source line number at next statement}  
 407 including: boolean; {no LHM's for included code}  
 408 eofexpected: 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 fptr: ip; {head of chain of forward reference pointers}  
 415 progid: ip; {program identifier}  
 416 curproc: 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: {str for current file name}  
 423 (arrays)  
 424 source: ftype; {name of pascal source file}  
 425 strbuf: array[1..max] of char;  
 426 lop: array[boolean] of ip; {fname: standard input, true: standard output}  
 427 rv: array[rrange] of alpha; {reserved words}  
 428 frv: array[0..ldim] of integer; {reserved words}  
 429 (indices in rv)  
 430 rsv: array[rrange] of symbol; {symbol for reserved words}  
 431 (arrays)  
 432 ch: array[chr] of character; {char type of a character}  
 433 chtype: (ch: character)  
 434 sy: array[rrange] of symbol; {symbol for single character symbols}  
 435 lms: array[lldim] of ip; {lms: array[1..4] of char; {mnemonics of pascal library routines}  
 436 opt: array['a'..'z'] of integer;  
 437 foroopt: array['a'..'z'] of boolean; {for different options}  
 438 undefip: array[idclass] of ip; {used in searchid}  
 439 (arrays)  
 440 rrv: array[0..maxrg] of record name: alpha; ad: integer end; {here here the external heading names}  
 441 (files)

```

449  ml:file of byte;  (the M1 code)
450  errors:file of error;  (the compilation errors)
451  (=====)
452  procedure gen2bytes(b:byte; i:integer);
453  var b1,b2:byte;
454  begin
455  if i<0 then
456  if i<minint then begin b1:=0; b2:=7 end
457  else begin i:=i-1; b1:=tda1 - i mod td; b2:=tda1 - i div td end
458  else begin b1:=i mod td; b2:=i div td end;
459  write(ml,b1,b2)
460  end;
461
462  procedure genct(i:integer);
463  begin
464  if (i>0) and (i<sp_max0) then write(ml,i,sp_max0)
465  else gen2bytes(sp_max0,i)
466  end;
467
468  procedure genclb(i:integer);
469  begin if i<td then write(ml,sp_1lb1,i) else gen2bytes(sp_1lb2,i) end;
470
471  procedure genlilb(i:integer);
472  begin lino:=lino+1;
473  if i<sp_max0 then write(ml,i,sp_max0) else genclb(i);
474  end;
475
476  procedure genclb1(i:integer);
477  begin if i<td then write(ml,sp_d1b1,i) else gen2bytes(sp_d1b2,i) end;
478
479  procedure gen0(b:byte);
480  begin write(ml,b); lino:=lino+1 end;
481
482  procedure gen1(b:byte; i:integer);
483  begin gen0(b); genct(i) end;
484
485  procedure gen2(b:byte; d:integer);
486  begin gen0(b); genclb(d) end;
487
488  procedure genident(name:type; var a:alpha);
489  var i,j:integer;
490  begin i:=tdmax;
491  while (a[i]=' ') and (i>1) do i:=i-1;
492  write(ml,name,i);
493  for j:=1 to i do write(ml,ord(a[j]));
494  end;
495
496  procedure genmp(m:integer);
497  var i:integer;
498  begin gen0(sp_max); write(ml,sp_max,m);
499  for i:=1 to m do write(ml,ord(linml[i]));
500  end;
501
502  procedure genman(b:byte; fil:ip);

```

```

505  var n:alpha; i,j:integer;
506  begin
507  if fil.p_pos.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_max,n)
513  end;
514
515  procedure genend;
516  begin write(ml,sp_max) end;
517
518  procedure genlin;
519  begin give_lino:=false;
520  if opt['l']<off then if main then gen1(sp_max,orig)
521  end;
522
523  procedure genreg(ad,az,nr:integer);
524  begin
525  if az<wordsize then
526  begin gen1(sp_max,mesreg); genct(ad); genct(nr); genend end
527  end;
528
529  (=====)
530
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  begin e:=err; write(errors,e);
535  if err>0 then begin gen1(sp_max,meserror); genend end
536  end;
537
538  procedure error(err:integer);
539  begin e:=spaces; e:=e-1; puterr(err) end;
540
541  procedure errid(err:integer; var id:alpha);
542  begin e:=id; e:=e-1; puterr(err) end;
543
544  procedure errint(err:integer; i:integer);
545  begin e:=i; e:=spaces; puterr(err) end;
546
547  procedure asper(err:integer);
548  begin if e.sp<nil then begin error(err); e.sp:=nil end end;
549
550  procedure teststand;
551  begin if opt<off then error(-e01) end;
552
553  procedure enterid(fil:ip);
554  (enter id pointed at by fil into the name-table,
555  which on each declaration level is organized as
556  an unbalanced binary tree)
557  var n:alpha; lip,lip1:ip; lleft,again:boolean;
558  begin n:=fil.p_name; again:=false;
559  lip:=top.p_fname;
560

```

```

561  if lip=nil then top.p_fname:=fil else
562  begin
563  repeat lip:=lip1;
564  if lip.p_name<n then
565  begin lip:=lip.p_llink; lleft:=true end
566  else
567  begin if lip.p_name=n 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;
572  end;
573  lip:=lip1; lip:=lip1; lip:=lip1;
574  if again then error(-e02,n);
575  end;
576
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;
583
584  procedure initf(fil:ip; fd:integer);
585  begin with a do begin
586  asp:=fil.p_pos; p:=fd; ak:=fixed; pos.ad:=fd; pos.lv:=level;
587  #ifdef SEGMENTS
588  pos.ad:=0;
589  #endif
590  end end;
591
592  function newp(kl:ideclass; n:alpha; id:top; actip:ip);
593  var p:ip; fil:flagst;
594  begin fil:=[];
595  case kl of
596  types,corrbod: (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 new(p,field); p.p_offset:=0 end;
604  proc_func: (same structure)
605  begin new(p,proc_act); 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_class:=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;
612
613  function newp(sf:structform; sz:integer);
614  var p:ip; sf:flagst;
615  begin sf:=[];
616  case sf of

```

```

617  scalar:
618  begin new(p,scalar); p.p_scalar:=0; p.p_const:=nil end;
619  subrange:
620  new(p,subrange);
621  pointer:
622  begin new(p,pointer); p.p_ctype:=nil end;
623  power:
624  new(p,power);
625  files:
626  begin new(p,files); sf:=[]; (with file) end;
627  arrays,corray: (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_fno:=sf; p.p_size:=sz; p.p_sf:=sf; newp:=p;
637
638  procedure init;
639  var a:char;
640  begin
641  (initialize the first name space)
642  new(top,blk); top.p_occure:=ak; top.p_rlink:=nil; top.p_fname:=nil;
643  level:=0;
644  (reserved words)
645  rw[0]:='if'; rw[1]:='do'; rw[2]:='of';
646  rw[3]:='to'; rw[4]:='in'; rw[5]:='on';
647  rw[6]:='and'; rw[7]:='for'; rw[8]:='nil';
648  rw[9]:='var'; rw[10]:='div'; rw[11]:='mod';
649  rw[12]:='set'; rw[13]:='and'; rw[14]:='not';
650  rw[15]:='then'; rw[16]:='else'; rw[17]:='with';
651  rw[18]:='case'; rw[19]:='type'; rw[20]:='goto';
652  rw[21]:='file'; rw[22]:='begin'; rw[23]:='until';
653  rw[24]:='while'; rw[25]:='array'; rw[26]:='const';
654  rw[27]:='label'; rw[28]:='repeat'; rw[29]:='record';
655  rw[30]:='down to'; rw[31]:='packed'; rw[32]:='program';
656  rw[33]:='function'; rw[34]:='procedure';
657  (corresponding symbols)
658  rsf[0]:='ifay'; rsf[1]:='doay'; rsf[2]:='ofay';
659  rsf[3]:='toay'; rsf[4]:='inay'; rsf[5]:='onay';
660  rsf[6]:='anday'; rsf[7]:='foray'; rsf[8]:='nilay';
661  rsf[9]:='varay'; rsf[10]:='divay'; rsf[11]:='moday';
662  rsf[12]:='setay'; rsf[13]:='anday'; rsf[14]:='notay';
663  rsf[15]:='thenay'; rsf[16]:='elseay'; rsf[17]:='withay';
664  rsf[18]:='caseay'; rsf[19]:='typeay'; rsf[20]:='gotay';
665  rsf[21]:='fileay'; rsf[22]:='beginay'; rsf[23]:='untilay';
666  rsf[24]:='whileay'; rsf[25]:='arrayay'; rsf[26]:='constay';
667  rsf[27]:='labelay'; rsf[28]:='repeatay'; rsf[29]:='recorday';
668  rsf[30]:='down toay'; rsf[31]:='packeday'; rsf[32]:='programay';
669  rsf[33]:='functay'; rsf[34]:='proceduray';
670
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(';')::=scary;
687   os(',')::=apunctoh; os(';')::=rparantoh; os(';')::=scary;
688   os(':')::=pericoh;   os(';')::=scamach;   os(';')::=minoh;
689   os(';')::=smich;     os('<')::=dash;     os(';')::=oolonoh;
690   os(';')::=grateroh; os(';')::=lbrackoh; os(';')::=equal;
691   os(';')::=arrouh;   os(';')::=lbrackoh; os(';')::=rbrackoh;
692   (single character symbols in char type order)
693   os(';')::=parantoh; os(';')::=rparantoh; os(';')::=scamach;
694   os(';')::=lbrackoh; os(';')::=rbrackoh; os(';')::=scamach;
695   os(';')::=smicoloh; os(';')::=arrowh; os(';')::=arrow;
696   os(';')::=plush;   os(';')::=dash;   os(';')::=dash;
697   os(';')::=dash;   os(';')::=dash;   os(';')::=dash;
698   os(';')::=dash;   os(';')::=dash;   os(';')::=dash;
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.intsize);
710   longptr :=newip(scalar.intsize);
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,all); 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   rtype:=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[page] :='page';
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[ord] :='ord';
746   p[abs] :='abs'; p[abs] :='abs';
747   p[ord] :='ord'; p[ord] :='ord';
748   p[ord] :='ord'; p[ord] :='ord';
749   p[ord] :='ord'; p[ord] :='ord';
750   p[ord] :='ord'; p[ord] :='ord';
751   p[ord] :='ord'; p[ord] :='ord';
752   p[ord] :='ord'; p[ord] :='ord';
753   p[ord] :='ord'; p[ord] :='ord';
754   p[ord] :='ord'; p[ord] :='ord';
755   p[ord] :='ord'; p[ord] :='ord';
756   (parameter types of standard functions)
757   rtype[foef] :=nil; rtype[foefin] :=nil;
758   rtype[foah] :=nil; rtype[foah] :=nil;
759   rtype[foah] :=nil; rtype[foah] :=nil;
760   rtype[foah] :=nil; rtype[foah] :=nil;
761   rtype[foah] :=nil; rtype[foah] :=nil;
762   rtype[foah] :=nil; rtype[foah] :=nil;
763   rtype[foah] :=nil; rtype[foah] :=nil;
764   rtype[foah] :=nil; rtype[foah] :=nil;
765   rtype[foah] :=nil; rtype[foah] :=nil;
766   (standard procedure/function identifiers)
767   for j:=read to halt 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 farotan do
772   begin new(p,func,standpf); p^.klass:=func; p^.idtype:=rtype[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   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[OPW] :='opw'; lnn[GRX] :='grx'; lnn[RDI] :='rdi';
788   lnn[RDC] :='rdc'; lnn[RDE] :='rde'; lnn[RDL] :='rdl';
789   lnn[RLM] :='rlm'; lnn[PUI] :='pui'; lnn[URI] :='uri';
790   lnn[UM] :='um'; lnn[UC] :='uc'; lnn[USC] :='usc';
791   lnn[USI] :='usi'; lnn[US] :='us'; lnn[USB] :='usb';
792   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
793   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
794   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
795   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
796   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
797   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
798   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
799   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
800   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
801   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
802   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
803   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
804   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
805   lnn[US] :='us'; lnn[US] :='us'; lnn[US] :='us';
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']:=true;
812   opt['p']:=true;
813   opt['s']:=addresssize div wordsize; (default pointer size in words)
814   opt['t']:=true;
815   opt:=off;
816   (scalar variables)
817   b:=nil;
818   b:=nil;
819   b:=nil;
820   b:=nil;
821   b:=nil;
822   c:=nil;
823   c:=nil;
824   c:=nil;
825   c:=nil;
826   e:=nil;
827   e:=nil;
828   l:=0;
829   d:=0;
830   s:=0;
831   l:=0;
832   g:=nil;
833   g:=nil;
834   w:=nil;
835   f:=nil;
836   f:=nil;
837   s:=nil;
838   s:=nil;
839   s:=nil;
840   s:=nil;

```

```

841   argv[0].ed:=1;
842   end;

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

861   (=====)
862   procedure trace(tname:alpha; fip:ip; var nandib:integer);
863   var i:integer;
864   begin
865   if opt['t']<off then
866   begin
867   if nandib=0 then
868   begin
869   d:=d+1;
870   begin d:=d+1; nandib:=d; gen(d:=d);
871   gen0(p_mes,um); write(um,sp,ocn,8);
872   for i:=1 to 8 do write(um,ord(fip^.name[i])); genend;
873   end;
874   gen((sp,um,0)); gen0(p_mes,nandib);
875   gen0(sp,ocn); genidnt(sp_mes,tname);
876   end;
877   end;

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

880   function simeof(fsp:sp):integer;
881   var s:integer;
882   begin s:=0;
883   if fsp<nil then s:=fsp^.size;
884   if s<0 then if odd(s) then s:=s+1;
885   simeof:=s;
886   end;

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

889   procedure exchange(i1,i2:integer);
890   var d1,d2:integer;
891   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 atop(a:byte);
902   begin gen!(a,even(sizeof(a.asp))) end;
903
904   procedure spndemptxet(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_lae,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 decaddr(var p:position);
948   begin if p.lv=0 then gen!(op_lae,p.ad) else loadpos(p,ptrsize) end;
949
950   procedure loadaddr;
951   begin with a do begin
952     case ok of
```

```
953     fixed;
954     with pos do
955       if lv<0 then
956         #ifdef 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         p:=pos;
966         loadpos(pos,ptrsize);
967         p:=pos;
968         ;
969         indexed;
970         gen!(op_sas);
971         end; [case]
972         sk:=loaded;
973     end end;
974
975   procedure load;
976   var sz:integer;
977   begin with a do begin
978     sz:=sizeof(asp); if not packbit then sz:=seven(sz);
979     if asp=all then
980       case ok of
981         cat:
982           gen!(op_loc,pos,ad); [only one-word scalars]
983         fixed:
984           loadpos(pos,sz);
985         p:=pos;
986         #ifdef SECTIONS
987           begin loadpos(pos,ptrsize); gen!(op_loi,sz) end;
988         loaded:
989           ;
990         p:=pos;
991         indexed:
992           gen!(op_lsa);
993         end; [case]
994         sk:=loaded;
995     end end;
996
997   procedure store;
998   var sz:integer;
999   begin with a do begin
1000     sz:=sizeof(asp); if not packbit then sz:=seven(sz);
1001     if asp=all then
1002       case ok of
1003         fixed:
1004           with pos do
1005             if lv<0 then
1006               #ifdef 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   p:=pos;
1018   loadpos(pos,ptrsize); gen!(op_sti,sz) end;
1019   p:=pos;
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 loadaddr; gen!(op_mdi,off) end
1030   end;
1031
1032   procedure loadheap;
1033   begin if forsof(a.asp,[arrays..records]) then loadaddr else load end;
1034   [.....]
1035
1036   procedure nextch;
1037   begin
1038     col:=col+1; read(input,cb); e:=col; e:=col-1; ch:=col[cb];
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 f1used then begin gen!(ps_esc,seeffloat); gen!(d2) end;
1048             gen!(ps_esc,off)
1049           end;
1050     #ifdef STANDARD
1051     goto 3999
1052     #endif
1053   #ifdef STANDARD
1054   halt
1055   #endif
1056   end;
1057   e:=col; e:=col-1; e:=col; e:=col-1; e:=col; e:=col-1;
1058   if not including then
1059     begin e:=col; e:=col-1; e:=col; e:=col-1; e:=col; e:=col-1;
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:=ch 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=[0-9] then
1081               repeat i:=i*10 + ord(c) - ord('0'); goto;
1082                 until c=[0-9]
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 ch=[0-9] 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:=e+1; e:=e+1;
1106                     if j=1; if j<=e then e:=e+1; e:=e+1;
1107                   end;
1108             nextch
1109           end;
1110     end;
1111     if source=empty then source:=e; e:=e;
1112     including:=source<=e; e:=e;
1113     i:=i-1; e:=e+1;
1114     if not including then e:=e; e:=e;
1115     end;
1116     while not eof do nextch;
1117   end;
1118
1119   procedure putdig;
1120   begin i:=i+1; if i<=max 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   until ch<=digit;
1130   [lower=0,upper=1,digit=2, ugly but fast]
1131   for i:=frk[i]-1 to frk[i]-1 do
1132     if rw[i]id then
1133       begin sy:=ray[i]; goto 1 end;
1134   sy:=idmax;
1135   1:
1136   end;
1137
1138 procedure innumber;
1139 label 1;
1140 const lam = 10;
1141 var
1142   i:integer;
1143   is:packed array[1..lam] of char;
1144   begin ix:=0; sy:=idmax; val:=0;
1145   repeat putdig until ch<=digit;
1146   if (ch='.') or (ch='e') or (ch='E') then
1147     begin
1148       if ch='.' then
1149         begin putdig;
1150           if ch='.' then
1151             begin seconddot:=true; ix:=ix+1; goto 1 end;
1152           if ch<=digit then error(+05) else
1153             repeat putdig until ch<=digit;
1154           end;
1155         if (ch='e') or (ch='E') then
1156           begin putdig;
1157             if (ch='e') or (ch='E') then putdig;
1158             if ch<=digit then error(+06) else
1159               repeat putdig until ch<=digit;
1160             end;
1161             if ix>lam then begin error(+07); ix:=lam end;
1162             sy:=word; frk:=frk; dlim:=nilbno+1; val:=nilbno;
1163             genidb(dlim); gen0(pm_rm); write(am,sp,room,ix);
1164             for i:=1 to ix do write(am,ord(strbuf[i])); genend;
1165             end;
1166           1:if (ch>=lower) or (ch>=upper) then teststandard;
1167           if sy=idmax then
1168             if ix>lam then error(+08) else
1169               begin ix:=0000000000; i:=lam+1;
1170                 while ix>0 do
1171                   begin i:=i-1; if i:=strbuf[ix]; ix:=ix-1 end;
1172                   if ix<=lam then
1173                     while i<=lam do
1174                       begin val:=val*10 + ord('0') + ord(strbuf[i]); i:=i+1 end
1175                     else if (ix<=lam) then (dopt:=off) then
1176                       begin sy:=longest; dlim:=dlim+1; val:=dlim;

```

```

1177   genidb(dlim); gen0(pm_rm); write(am,sp,room,ix+1-1);
1178   while ix<=lam do
1179     begin write(am,ord(strbuf[ix])); i:=i+1 end;
1180   genend;
1181   end error(+09);
1182   end;
1183   end;
1184
1185 procedure instrng(q:char);
1186 var i:integer;
1187 begin ix:=0; zerostring:=q;
1188 repeat
1189   repeat nextch; ix:=ix+1; if ix<=max then strbuf[ix]:=ch;
1190   until (ch=q) or eol;
1191   if ch=q then nextch else error(+10);
1192   until ch<=q;
1193   if not zerostring then
1194     begin ix:=ix-1; if ix=0 then error(+011) end
1195   else
1196     begin strbuf[ix]:=chr(0); if opt=off then error(+012) end;
1197     if (ix=1) and not zerostring then
1198       begin sy:=chr(ord); val:=ord(strbuf[1]) end
1199     else
1200       begin sy:=stringent; dlim:=dlim+1; val:=dlim;
1201         if ix>max then begin error(+013); ix:=max end;
1202         genidb(dlim); gen0(pm_rm); write(am,sp,room,ix);
1203         for i:=1 to ix do write(am,ord(strbuf[i])); genend;
1204       end;
1205   end;
1206
1207 procedure incomment;
1208 var stop:char;
1209 begin nextch; stop:='';
1210 if ch='*' then options:=true;
1211 while (ch<>'') and (ch<=stop) do
1212   begin stop:=''; if ch='*' then stop:='';
1213   if ch='*' then error(+014);
1214   if eol then nextch; nextch;
1215   end;
1216 if ch='*' then teststandard;
1217 nextch;
1218 end;
1219
1220 procedure inasy;
1221 read next basic symbol of source program and return its
1222 description in the global variables sy, op, id, val and ix
1223 label 1;
1224 begin
1225   1:case ch of
1226     ' ':begin
1227       begin e.chno:=e.chno - e.chno mod 8 + 8; nextch; goto 1 end;
1228     layout:
1229       begin if eol then nextch; nextch; goto 1 end;
1230     lower,upper: indent;
1231     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,mulch,slash,star,equal;
1262 begin sy:=asy(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 (case)
1269 end;
1270
1271 procedure nextf(fsy:symbol; err:integer);
1272 begin if sy=fsy then inasy else error(-err) end;
1273
1274 function find1(sy1,sy2:sos; err:integer):boolean;
1275 (symbol of sy1 expected. return true if sy in sy1)
1276 begin
1277   if not (sy in sy1) then
1278     begin error(-err); while not (sy in sy1+sy2) do inasy end;
1279   find1:=sy in sy1;
1280 end;
1281
1282 function find2(sy1,sy2:sos; err:integer):boolean;
1283 (symbol of sy1+sy2 expected. return true if sy in sy1)
1284 begin
1285   if not (sy in sy1+sy2) then
1286     begin error(-err); repeat inasy until sy in sy1+sy2 end;
1287   find2:=sy in sy1;
1288 end;

```

```

1289 end;
1290
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 inasy until sy in [sy1]+sy2 end;
1296   if sy=sy1 then inasy else find3:=false;
1297 end;
1298
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;
1304
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;
1310
1311 {.....}
1312 function searchid(fidals: setof id):ip;
1313 (search for current identifier symbol in the name table)
1314 label 1;
1315 var lip:ip; is:doless;
1316 begin lastap:=stop;
1317 while lastap<=nil do
1318   begin lip:=lastap+.fname;
1319     while lip<=nil do
1320       if lip<=nil then
1321         if lip<=nil then
1322           begin
1323             if lip<=nil then
1324               if lip<=nil then
1325                 lip:=lip+.lflg+.lflg+.lflg;
1326               goto 1;
1327             end
1328             else lip:=lip+.rlink;
1329             else
1330               if lip<=nil then lip:=lip+.rlink else lip:=lip+.lflg;
1331             lastap:=lastap+.alink;
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;
1343
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 1: 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 1: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_of; asp:=realptr; fltused:=true end;
1369 ri: begin op:=op_of; asp:=intptr; fltused:=true end;
1370 il: begin op:=op_of; asp:=longptr end;
1371 li: begin op:=op_of; asp:=intptr end;
1372 lr: begin op:=op_of; asp:=realptr; fltused:=true end;
1373 rl: begin op:=op_of; 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=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
1389 function desub(flpsp:sp);
1390 begin
1391 if formof(flps,subrange) then flps:=flps.rangetype; desub:=flps
1392 end;
1393
1394 function nicescalar(flpsp:sp):boolean;
1395 begin
1396 if flps=ll then nicescalar:=true else
1397 nicescalar:=(flps.for=scalar) and (flps<=realptr) and (flps<=longptr)
1398 end;
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=ntpr) and (q=realptr) then compat:=ir else
1466 if (p=realptr) and (q=intptr) then compat:=ri else
1467 if (p=intptr) and (q=longptr) then compat:=il else
1468 if (p=longptr) and (q=intptr) then compat:=li else
1469 if (p=longptr) and (q=realptr) then compat:=lr else
1470 if (p=realptr) and (q=longptr) then compat:=rl else
1471 ;
1472 pointer:
1473 if (p=ntpr) or (q=ntpr) then compat:=eq;
1474 power:
1475 if p=mpmptset then compat:=se else
1476 if q=mpmptset then compat:=se else
1477 if compat(p.elset,q.elset) <= subeq then
1478 compat:=subeq;
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 procedure checkasp(flpsp: err:integer);
1487 var ts:twostrut;
1488 begin
1489 ts:=compat(a.asp,flps);
1490 case ts of
1491 eq:
1492 if flps<=all then if withfile in flps.sflag then aspperr(err);
1493 subeq:
1494 checkbada(flps);
1495 li:
1496 begin oponent(ts); checkasp(flps,err) end;
1497 ll,rl,lr,lr:
1498 oponent(ts);
1499 oponent(ts);
1500 asp:=emptypset(flps);
1501 notaq,ri,se:
1502 aspperr(err);
1503 end;
1504 end;
1505
1506 procedure force(flpsp: err:integer);
1507 begin load; checkasp(flps,err) end;
1508
1509 function neident(kl:ld:laa; id:sp; nst:ip; err:integer):lp;
1510 begin neident:=null;
1511 if sp<=all then error(err) else
1512
```

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

```
1513 begin neident:=newip(kl.id,ld,ext); inay end
1514 end;
1515
1516 function stringstrut:sp;
1517 var lpsp:sp;
1518 begin (only used when ix and zerostring are still valid)
1519 if zerostring then lpsp:=stringptr else
1520 begin lpsp:=newip(array,ifcharize); lpsp.sflag:=speak;
1521 lpsp.selttype:=charptr; lpsp.instype:=null;
1522 end;
1523 stringstrut:=lpsp;
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(flpsp: pack:boolean):integer;
1541 var sz,min,max,tot,n:integer;
1542 begin sz:=sizeof(flps.selttype);
1543 if not pack then sz:=sz+1;
1544 if bounds(flps.instype,min,max) then (we checked before)
1545 dibo:=dibo+1; flps.arpos.lv:=0; flps.arpos.ed:=dibo;
1546 genid(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:lp);
1554 var lpsp:lp; i:integer;
1555 begin
1556 if flps<=all then
1557 begin treealk(flps.llink); treealk(flps.rlink);
1558 if flps.klans=vars then
1559 begin if not (used in flps.iflag) then errid(-+019),flps.name;
1560 if not (assigned in flps.iflag) then errid(-+020),flps.name;
1561 lpsp:=flps.idtype;
1562 if not (sorg in flps.iflag) then
1563 genreg(flps.vpos.ed,ssizeof(lpsp,ord(formof(lpsp,pointer)))));
1564 if flps<=all then if withfile in lpsp.sflag then
1565 if lpsp.for=files then
1566 if level=1 then
1567 begin
1568 for i:=2 to argo do with argo[i] do
```





