Pascal Users Group

NUMBER 22 & 23

Pascal News

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCALERS

SEPTEMBER, 1981

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

Return postage guaranteed
Address Correction requested

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.
	[ ] 2 years	$18.	#10.	DM45.	A$15.
[ ] Renew my subscription for:	[ ] 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 Pascal News.

[ ] 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

APPLICATION FOR LICENSE TO USE VALIDATION SUITE FOR PASCAL

Name and address of reqestor: _____

(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.

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

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

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 Pensylvanians 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 Hughes will be handling all business for Britain, and Hellmut Weber 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 donated 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.

THIS ISSUE

The highlight of ths 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 Gregg 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 probing article by Jonathan Yavner concerning problems with Pascal and some proposals for their solution.

Hope you like it.

Rick

Here and There With Pascal

Summary of **Implementations**

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 Intruments 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	

EM1 COMPILER


```
#include "../h/local.h"
#include "../h/em1.h"

{ (c) copyright 1980 by the Vrije Universiteit, Amsterdam, The Nether-
  lands.  Explicit permission is hereby granted to universities to use
  or duplicate this program for educational or research purposes.  All
  other  use or duplication  by universities,  and all use or duplica-
  tion by other organizations is expressly prohibited  unless  written
  permission  has  been obtained from the Vrije Universiteit. Requests
  for such permissions may be sent to

        Dr. Andrew S. Tanenbaum
        Wiskundig Seminarium
        Vrije Universiteit
        Postbox 7161
        1007 MC Amsterdam
        The Netherlands

  Organizations wishing to modify part of this software for subsequent
  sale  must  explicitly  apply  for  permission.  The exact arrange-
  ments will be worked out on a case by case basis, but at  a  minimum
  will require the organization to include the following notice in all
  software and documentation based on our work:

        This  product  is  based  on  the  Pascal  system
      developed  by  Andrew  S.  Tanenbaum, Johan W. Stevenson
      and Hans van Staveren of the Vrije  Universiteit,  Amster-
      dam, The Netherlands.
}

{if next line is included the compiler is written in standard pascal}
{#define       STANDARD        1}

{if next line is included, then code is produced for segmented memory}
{#define       SEGMENTS        1}

{Author:        Johan Stevenson                 Version:        31}
{$1- : no source line numbers}
{$r- : no subrange checking}
{$a- : no assertion checking}
#ifdef STANDARD
{$s+ : test conformancy to standard}
#endif

program pem(input,em1,errors);
{ This Pascal compiler produces EM1 code as described in
  - A.S.Tanenbaum, J.W.Stevenson & H. van Staveren,
        "Description of a experimental machine architecture for use of
        block structured languages" Informatika rapport 54,
  A description of Pascal is given in
  - K.Jensen & N.Wirth,PASCAL user manual and report,Springer-Verlag.
  Several options may be given in the normal pascal way. Moreover,
  a positive number may be used instead of + and -. The options are:
        e:      interpret assertions (+)
        o:      C-type strings allowed (-)
        d:      type long may be used (-)
        f:      size of reals in words (2)
        i:      controls the number of bits in integer sets (16)
        l:      insert code to keep track of source lines (+)
        o:      optimize (+)
        p:      size of pointers in words (1)
        r:      check subranges (+)
        s:      accept only standard pascal programs (-)
        t:      trace procedure entry and exit (-)
        u:      treat '_' as letter (-)
}
{*************************************************************}
#ifdef STANDARD
label 9999;
#endif

const

{powers of two}
  t7            = 128;
  t8m1          = 255;
  t8            = 256;
  t14           = 16384;
  t15m1         = 32767;

{EM-1 sizes}
  bytebits      = 8;
  wordbits      = 16;
  wbm1          = 15;           {wordbits-1}
  minint        = -t15m1;
  maxint        = t15m1;
  maxintstring  = '0000032767';
  maxlongstring = '2147483647';

  bytesize      = 1;
  wordsize      = 2;
  addrsize      = wordsize;
  pnumsize      = wordsize;
  shortsize     = wordsize;
  longsize      = 4;
#ifdef SFLOAT
  floatsize     = 4;
#endif
#ifndef SFLOAT
  floatsize     = 8;
#endif

{Pascal sizes, for ptrsize, realsize and fhsize see handleopts}
  { EM-1 requires that objects greater than a single byte start at a
    word  boundary,  so their address is even.  Normally, a full word
    is also allocated for objects of a single byte.  This extra  byte
    is really allocated to the object, not only skipped by alignment,
    i.e. if the value false is assigned to a  boolean  variable  then
    both bytes are cleared.  For single byte objects is packed arrays
    or packed records, however, only one byte is allocated,  even  if
    the next byte is unused.  Strings are packed arrays.  The size of
    pointers is 2 by default, but can be changed at  runtime  by  the
    p-option.  Floating point numbers in EM-1 currently have size 4,
    but this might change in the future to 8.  The default can be
    overwritten  by  the f-option.  The routines involved with align-
    ment are 'even', 'address' and 'arraysize'.
  }
  boolsize      = bytesize;
  charsize      = bytesize;
  intsize       = shortsize;
  buffsize      = 512;
  maxsetsize    = 4096;         {t15 div bytebits}

{maximal indices}
  idmax         = 8;
  fnmax         = 14;
  smax          = 72;
  rmax          = 72;
  imax          = 10;

{opt values}
  off           = 0;
  on            = 1;

{for push and pop: }
  global        = false;
  local         = true;

{set bounds}
  minsetlst     = 0;
  maxsetlst     = 15;           {default}

{constants describing the compact EM1 code}
  MAGICLOW      = 172;
  MAGICHIGH     = 0;
  memerror      = 0;
  mesoptoff     = 1;
  mesvirtual    = 2;
  mesmreg       = 3;
  mesline       = 4;
  mesfloats     = 5;

{ASCII characters}
  tab           = 9;
  newline       = 10;
  hortab        = 11;
  formfeed      = 12;
  carret        = 13;

{miscellaneous}
  maxsg         = 127;          {maximal segment number}
  maxcharord    = 127;          {maximal ordinal number of chars}
  maxargc       = 13;           {maximal index in argv}
  rwlim         = 38;           {number of reserved words}
  spaces        = '         ';
  emptyform     = '           ';

{-------------------------------------------------------------}
```



```
type
{scalar types}
  symbols       (comma,semicolon,colon1,colon2,notsy,lbrack,ident,
                 intcst,charcst,realcst,longcst,stringcst,nilcst,minsy,
                 plussy,lparent,arrow,arraysy,recordsy,setsy,filesy,
                 packedsy,progsy,labelsy,constsy,typesy,varsy,procsy,
                 funcsy,beginsy,gotosy,ifsy,whilesy,repeatsy,forsy,
                 withsy,casesy,becomes,starsy,divsy,modsy,slashsy,
                 andsy,orsy,eqsy,neqsy,gtsy,geqsy,ltsy,
                 leqsy,insy,endsy,elsesy,untilsy,ofsy,dosy,
                 downtosy,tosy,thensy,rbrack,rparent,period
                );              {the order is important}
  chartypes     (lower,upper,digit,layout,tabch,
                   quotech,dquotech,colonch,periodch,lessch,
                   greaterch,lparentch,lbracech,
                                        {different entries}
                 rparentch,lbrackch,rbrackch,commach,semich,arrowch,
                   plusch,minch,slash,star,equal,
                                        {also symbols}
                 others
                );
  standpf=      (pread,preadln,pwrite,pwriteln,pput,pget,
                 preset,prewrite,pnew,pdispose,ppack,punpack,
                 pmark,prelease,ppage,phalt,
                                        {all procedures}
                 feof,feoln,fabs,fsqr,ford,fchr,fpred,fsucc,fodd,
                 ftrunc,fround,fsin,fcos,fexp,fsqrt,fln,farctan
                                        {all functions}
                );              {the order is important}
  libnames=     (ELH ,EFL ,CLS ,WDW ,           {input and output}
                 OPN ,GETX,RDI ,RDC ,RDR ,RBL ,RLE ,
                                        {on inputfiles}
                 CRE ,PUTX,WRI ,WBI ,WRC ,WSC ,WRS ,WSS ,WRB ,
                 WBB ,WRR ,WBR ,WRL, WSL, WRF ,WRZ ,WSZ ,WLB ,PAG ,
                                {on outputfiles, order important}
                 ABR ,RND ,SIN ,COS ,EXPX,SQT ,LOG ,ATN ,
                                        {floating point}
                 ABI ,ABL ,BCP ,BTS ,HEWX,SAV ,RST ,INI ,MLT ,
                 ASS ,GTO ,PAC ,WBP, DIS, ABZ, MDI, MDL
                                        {miscellaneous}
                );
  structforms=  (scalar,subrange,pointer,power,files,arrays,carray,
                 records,variant,tag);          {order important}
  structflags=  (speak,withfile);
  identflags=   (refer,used,assigned,noreg,semeset);
  idclass=      (types,konst,vars,field,carrband,proc,func);
  kindofpfs=    (standard,formal,actual,extra,forward);
  where=        (blck,rec,wreo);
  strkind=      (cst,fixed,pfixed,loaded,ploaded,indexed);
  tmpstruct=    (eq,subeq,ir,ri,il,li,lr,rl,es,se,noteq);
                                        {order important}

\ {subrange types}
    sgrange=    0..maxsg;
    idrange=    1..idmax;
    fnrange=    1..fnmax;
```

```
    rwrange=      0..rwlim;
    bytes=        0..tbm1;

  {pointer types}
    sp=   ^structure;
    ip=   ^identifier;
    lp=   ^lbl;
    bp=   ^blockinfo;
    np=   ^nameinfo;

  {set types}
    sos=          set of symbol;
    setofids=     set of idclass;
    formset=      set of structform;
    sflagset=     set of structflag;
    iflagset=     set of identflag;

  {array types}
    alpha =packed array[idrange] of char;
    fntype=packed array[farange] of char;

  {record types}
    errec=record
      errno:integer;                {error number}
      mess:alpha;                   {identifier parameter if required}
      mesi:integer;                 {numeric parameter if required}
      chno:integer;                 {column number}
      linc:integer;                 {line number}
      linr:integer;                 {relative to start of (included) file}
      orig:integer;                 {idem, but before preprocessing}
      fnam:fntype;                  {source file name}
    end;

    position=record                 {the addr info of certain variable}
      ad:integer;                   {for locals it is the byte offset}
      lv:integer;                   {the level of the beast}
#ifdef SEGMENTS
      sg:segrange                   {only relevant for globals (lv=0) }
#endif
    end;

  {records of type attr are used to remember qualities of
    expression parts to delay the loading of them.
    Reasons to delay the loading of one word constants:
          - bound checking
          - set building.
    Reasons to delay the loading of direct accessible objects:
          - efficient handling of read/write
          - efficient handling of the with statement.
  }
    attr=record
      exp:sp;                           {type of expression}
      packbit:boolean;                  {true for packed elements}
      ak:attrkind;                      {access method}
      pos:position;                     {sg, lv and ad}
      {if ak=cst then the value is stored in ad}
    end;

    nameinfo=record                 {one for each separate name space}
      nlink:np;                     {one deeper}
      fname:ip;                     {first name: root of tree}
      case occur:where of
        blok:();
        rec: ();
        wrec:(wa:attr)              {name space opened by with statement}
    end;

    blockinfo=record        {all info of the current procedure}
      nextbp:bp;            {pointer to blockinfo of surrounding proc}
      lc:integer;           {data location counter (from begin of proc) }
      ilbno:integer;        {number of last local label}
      forwcount:integer;    {number of not yet specified forward procs}
      lchain:lp;            {first label: header of chain}
    end;

    structure=record
      size:integer;                             {size of structure in bytes}
      sflag:sflagset;                           {flag bits}
      case form:structform of
        scalar  :(scalno:integer;               {number of range descriptor}
                  fconst:ip                     {names of constants}
                 );
        subrange:(min,max:integer;              {lower and upper bound}
                  rangetype:sp;                 {type of bounds}
                  subrno:integer                {number of subr descriptor}
                 );
        pointer :(eltype:sp);                   {type of pointed object}
        power   :(elset:sp);                    {type of set elements}
        files   :(filtype:sp);                  {type of file elements}
        arrays,carray:
                 (eltype:sp;                    {type of array elements}
                  inxtype:sp;                   {type of array index}
                  arpos:position                {position of array descriptor}
                 );
        records :(fstfld:ip;                    {points to first field}
                  tagsp:sp                      {points to tag if present}
                 );
        variant :(varval:integer;               {tag value for this variant}
                  nxtvar:sp;                    {next equilevel variant}
                  subtsp:sp                     {points to tag for sub-case}
                 );
        tag     :(fstvar:sp;                    {first variant of case}
                  tfldsp:sp                     {type of tag}
                 )
    end;

    identifier=record
      idtype:sp;                            {type of identifier}
      name:alpha;                           {name of identifier}
      llink,rlink:ip;                       {see enterid,searchid}
      next:ip;                              {used to make several chains}
      iflag:iflagset;                       {several flag bits}
```



```
    case klass:idclass of
      types   :();
      konst   :(value:integer);         {for integers the value is
                                         computed and stored in this field.
                                         For strings and reals an assembler constant is
                                         defined labeled '.1', '.2', ...
                                         This '.' number is then stored in value.
                                         For reals value may be negated to indicate that
                                         the opposite of the assembler constant is needed. }
      vars    :(vpos:position);         {position of var}
      field   :(foffset:integer);       {offset to begin of record}
      carrbnd :();                      {idtype points to carray}
      proc,func:
        (case pfkind:kindofpf of
           standard:(key:standpf);      {identification}
           formal,actual,forward,extrn:
             (pfpos:position;           {lv gives declaration level.
                                         sg gives instruction segment of this proc and
                                         ad is relevant for formal pf's and for
                                         functions (no conflict!!).
                                         for functions: ad is the result address.
                                         for formal pf's: ad is the address of the
                                         descriptor }
              pfno:integer;             {unique pf number}
              parhead:ip;               {head of parameter list}
              handle:integer            {le when heading scanned}
             )
        )
  end;

  lblrecord
    nextlp:lp;              {chain of labels}
    seen:boolean;
    lbval:integer;          {label number given by the programmer}
    lbname:integer;         {label number given by the compiler}
    lbdlb:integer           {zero means only locally used,
                             otherwise dlbno of label information}
  end;

{------------------------------------------------------------------------}
var  {the most frequest used externals are declared first}
  sy:symbol;                {last symbol}
  a:attr;                   {type,access method,position,value of expr}
{returned by insym}
  ch:char;                  {last character}
  chsy:chartype;            {type of ch, used by insym}
  val:integer;              {if lsmt symbol is an constant }
  iz:integer;               {string length}
  eol:boolean;              {true if current ch replaces a newline}
  inrestring:boolean;       {true for strings in ' '}
  id:alpha;                 {if last symbol is an identifier}
{some counters}
  line:integer;             {line number on code file (1..n) }
  dlbno:integer;            {number of last global number}
  lmax:integer;             {keeps track of maximum of lc}
  level:integer;            {current static level}
  ptrsize:integer;
  realsize:integer;
  fhsize:integer;           {file header size}
  argc:integer;             {index in argv}
  lastpfno:integer;         {unique pf number counter}
  oopt:integer;             {C-type strings allowed if on}
  dopt:integer;             {longs allowed if on}
  iopt:integer;             {number of bits in sets with base integer}
  aopt:integer;             {standard option}
{pointers pointing to standard types}
  realptr,intptr,textptr,emptyset,boolptr:sp;
  charptr,nilptr,stringptr,longptr:sp;
{flags}
  giveline:boolean;         {give source line number at next statement}
  including:boolean;        {no LIN's for included code}
  eofexpected:boolean;      {quit without error if true (nextch) }
  main:boolean;             {complete programme or a module}
  intypedec:boolean;        {true if nested in typedefinition}
  fltused:boolean;          {true if floating point instructions are used}
  seconddot:boolean;        {indicates the second dot of '..'}
{pointers}
  fwptr:ip;                 {head of chain of forward reference pointers}
  progp:ip;                 {program identifier}
  currproc:ip;              {current proc/func ip (see casestatement)}
  top:np;                   {pointer to the most recent name space}
  lastap:np;                {pointer to nameinfo of last searched ident }
{records}
  b:blockinfo;              {all info to be stacked at pfdeclaration}
  e:errec;                  {all info required for error messages}
  fa:attr;                  {attr for current file name}
{arrays}
  source:fntype;            {name of pascal source file}
  strbuf:array[1..smax] of char;
  iop:array[boolean] of ip;
                            {false:standard input, true:standard output}
  rw:array[rwrange] of alpha;
                            {reserved words}
  frw:array[0..idmax] of integer;
                            {indices in rw}
  rsy:array[rwrange] of symbol;
                            {symbol for reserved words}
  os:array[char] of chartype;
                            {chartype of a character}
  ssy:array[rparentoh..equal] of symbol;
                            {symbol for single character symbols}
  lmn:array[libmnem] of packed array[1..4] of char;
                            {mnemonics of pascal library routines}
  opt:array['a'..'z'] of integer;
  forceopt:array['a'..'z'] of boolean;
                            {26 different options}
  undefip:array[idclass] of ip;
                            {used in searchid}
  argv:array[0..maxargs] of
        record name:alpha; ad:integer end;
                            {save here the external heading names}
{files}
```

```
449     em1:file of byte;     {the EM1 code}
450     errors:file of errec;
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:=t7 end
459       else begin i:=-i-1; b1:=t8m1 - i mod t8; b2:=t8m1 - i div t8 end
460     else begin b1:=i mod t8; b2:=i div t8 end;
461     write(em1,b,b1,b2)
462   end;

464   procedure genost(i:integer);
465   begin
466     if (i>=0) and (i<sp_nost0) then write(em1,i+sp_fost0)
467     else gen2bytes(sp_ost2,i)
468   end;

470   procedure genlb(i:integer);
471   begin if i<t8 then write(em1,sp_ilb1,i) else gen2bytes(sp_ilb2,i) end;

473   procedure genilb(i:integer);
474   begin lino:=lino-1;
475     if i<sp_nilb0 then write(em1,i+sp_filb0) else genlb(i);
476   end;

478   procedure gendlb(i:integer);
479   begin if i<t8 then write(em1,sp_dlb1,i) else gen2bytes(sp_dlb2,i) end;

481   procedure gen0(b:byte);
482   begin write(em1,b); lino:=lino+1 end;

484   procedure gen1(b:byte; i:integer);
485   begin gen0(b); genost(i) end;

487   procedure gend(b:byte; d:integer);
488   begin gen0(b); gendlb(d) end;

490   procedure genident(nametype:byte; var s:alpha);
491   var i,j:integer;
492   begin i:=idnam;
493     while (s[i]=' ') and (i>1) do i:=i-1;
494     write(em1,nametype,i);
495     for j:=1 to i do write(em1,ord(s[j]))
496   end;

498   procedure gensp(m:libsmnem);
499   var i:integer;
500   begin gen0(op_eal); write(em1,sp_pnam,4);
501     for i:=1 to 4 do write(em1,ord(lms[m][i]))
502   end;

504   procedure genpnam(b:byte; fip:ip);
505   var n:alpha; i,j:integer;
506   begin
507     if fip^.pfpos.lv<=1 then n:=fip^.name else
508       begin n:='          '; j:=1; i:=fip^.pfno;
509         while i<>0 do
510           begin j:=j+1; n[j]:=chr(i mod 10 + ord('0')); i:=i div 10 end;
511       end;
512     gen0(b); genident(sp_pnam,n)
513   end;

515   procedure genend;
516   begin write(em1,sp_cend) end;

518   procedure genlin;
519   begin giveline:=false;
520     if opt['l']<>off then if main then gen1(op_lin,e.orig)
521   end;

523   procedure genreg(ad,sz,nr:integer);
524   begin
525     if sz<=wordsize then
526       begin gen1(ps_mes,mesreg); genost(ad); genost(nr); genend end
527   end;

529   {*************************************************************}

531   procedure puterr(err:integer);
532   {as you will notice, all error numbers are preceded by '+' and '0' to
533     ease their renumbering in case of new errornumbers.
534   }
535   begin e.erno:=err; write(errors,e);
536     if err>0 then begin gen1(ps_mes,meserror); genend end
537   end;

539   procedure error(err:integer);
540   begin e.mess:=spaces; e.mesi:= -1; puterr(err) end;

542   procedure errid(err:integer; var id:alpha);
543   begin e.mess:=id; e.mesi:= -1; puterr(err) end;

545   procedure errint(err:integer; i:integer);
546   begin e.mesi:=i; e.mess:=spaces; puterr(err) end;

548   procedure asperr(err:integer);
549   begin if e.asp<>nil then begin error(err); e.asp:=nil end end;

551   procedure teststandard;
552   begin if sopt<>off then error(-(+01)) end;

554   procedure enterid(fip: ip);
555   {enter id pointed at by fip into the name-table,
556     which on each declaration level is organised as
557     an unbalanced binary tree}
558   var nam:alpha; lip,lip1:ip; lleft,again:boolean;
559   begin nam:=fip^.name; again:=false;
560     lip:=top^.fname;
```



```
561     if lip=nil then top^.fname:=fip else
562       begin
563         repeat lip1:=lip;
564           if lip^.name>nam then
565             begin lip:=lip^.llink; lleft:=true end
566           else
567             begin if lip^.name=nam then again:=true;  {name conflict}
568               lip:=lip^.rlink; lleft:=false;
569             end;
570         until lip=nil;
571         if lleft then lip1^.llink:=fip else lip1^.rlink:=fip
572       end;
573     fip^.llink:=nil; fip^.rlink:=nil;
574     if again then errid(+02,nam);
575   end;

577   procedure initpos(var p:position);
578   begin p.lv:=level; p.ad:=0;
579   {ifdef SEGMENTS
580     p.sg:=0
581   }endif
582   end;

584   procedure init1(fsp:sp; fad:integer);
585   begin with s do begin
586     sop:=fsp; packbit:=false; ak:=fixed; pos.ad:=fad; pos.lv:=level;
587   {ifdef SEGMENTS
588     pos.sg:=0;
589   }endif
590   end end;

592   function newip(kl:idclass; n:alpha; idt:tp; nxt:ip):ip;
593   var p:ip; f:iflagset;
594   begin f:=[];
595     case kl of
596       types,varbnd:  {similar structure}
597         new(p,types);
598       konst:
599         begin new(p,konst); p^.value:=0 end;
600       vars:
601         begin new(p,vars); f:=[used,assigned]; initpos(p^.vpos) end;
602       field:
603         begin new(p,field); p^.foffset:=0 end;
604       proc,func:  {same structure}
605         begin new(p,proc,actual); p^.pfkind:=actual;
606           initpos(p^.pfpos); p^.pfno:=0; p^.parhead:=nil; p^.headlc:=0
607         end
608     end;
609     p^.name:=n; p^.klass:=kl; p^.idtype:=idt; p^.next:=nxt;
610     p^.llink:=nil; p^.rlink:=nil; p^.iflag:=f; newip:=p
611   end;

613   function newsp(sf:structform; sz:integer):sp;
614   var p:sp; sflag:sflagset;
615   begin sflag:=[];
616     case sf of
617       scalar:
618         begin new(p,scalar); p^.scalno:=0; p^.fconst:=nil end;
619       subrange:
620         new(p,subrange);
621       pointer:
622         begin new(p,pointer); p^.eltype:=nil end;
623       power:
624         new(p,power);
625       files:
626         begin new(p,files); sflag:=[withfile] end;
627       arrays,carray:  {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^.form:=sf; p^.size:=sz; p^.sflag:=sflag; newsp:=p;
637   end;

639   procedure init1;
640   var c:char;
641   begin
642   {initialise the first name space}
643     new(top,blok); top^.occur:=blok; top^.nlink:=nil; top^.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]:='or      ';
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]:='downto  ';  rw[31]:='packed  ';  rw[32]:='program ';
657     rw[33]:='function';  rw[34]:='procedur';
658   {corresponding symbols}
659     rsy[ 0]:=ifsy;       rsy[ 1]:=dosy;       rsy[ 2]:=ofsy;
660     rsy[ 3]:=tosy;       rsy[ 4]:=insy;       rsy[ 5]:=orsy;
661     rsy[ 6]:=endsy;      rsy[ 7]:=forsy;      rsy[ 8]:=nilcst;
662     rsy[ 9]:=varsy;      rsy[10]:=divsy;      rsy[11]:=modsy;
663     rsy[12]:=setsy;      rsy[13]:=andsy;      rsy[14]:=notsy;
664     rsy[15]:=thensy;     rsy[16]:=elsesy;     rsy[17]:=withsy;
665     rsy[18]:=casesy;     rsy[19]:=typesy;     rsy[20]:=gotosy;
666     rsy[21]:=filesy;     rsy[22]:=beginsy;    rsy[23]:=untilsy;
667     rsy[24]:=whilesy;    rsy[25]:=arraysy;    rsy[26]:=constsy;
668     rsy[27]:=labelsy;    rsy[28]:=repeatsy;   rsy[29]:=recordsy;
669     rsy[30]:=downtosy;   rsy[31]:=packedsy;   rsy[32]:=progsy;
670     rsy[33]:=funcsy;     rsy[34]:=procsy;
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       frw[5]:=28; frw[6]:=32; frw[7]:=33; frw[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(newline)]:=layout;
680       os[chr(hortab)]:=layout;
681       os[chr(formfeed)]:=layout;
682       os[chr(carret)]:=layout;
683     {characters with corresponding chartype in ASCII order}
684       os[chr(tab)]:=tabch;
685       os[' ']:=layout;      os['"']:=dquotech;      os['''']:=quotech;
686       os['(']:=lparentch;   os[')']:=rparentch;     os['*']:=star;
687       os['+']:=plusch;      os[',']:=commach;       os['-']:=minch;
688       os['.']:=periodch;    os['/']:=slash;         os[':']:=colonch;
689       os[';']:=semich;      os['<']:=lessch;        os['=']:=equal;
690       os['>']:=greaterch;   os['[']:=lbrackch;      os[']']:=rbrackch;
691       os['^']:=arrowch;     os['{']:=lbracech;
692     {single character symbols in chartype order}
693       osy[rparentch]:=rparent;      osy[lbrackch]:=lbrack;
694       osy[rbrackch]:=rbrack;        osy[commach]:=comma;
695       osy[semich]:=semicolon;       osy[arrowch]:=arrow;
696       osy[plusch]:=plussy;          osy[minch]:=minsy;
697       osy[slash]:=slashsy;          osy[star]:=starsy;
698       osy[equal]:=eqsy;
699     end;

700
701     procedure init2;
702     var p,q:ip; k:idclass;
703     begin
704     {undefined identifier pointers used by searchid}
705       for k:=types to func do
706         undefip[k]:=newip(k,spaces,nil,nil);
707     {standard type pointers. some size are filled in by handleopts}
708       intptr   :=newsp(scalar,intsize);
709       realptr  :=newsp(scalar,0);
710       longptr  :=newsp(scalar,longsize);
711       charptr  :=newsp(scalar,charsize);
712       boolptr  :=newsp(scalar,boolsize);
713       nilptr   :=newsp(pointer,0);
714       stringptr:=newsp(pointer,0);
715       emptyset :=newsp(power,intsize); emptyset^.elset:=nil;
716       textptr  :=newsp(files,0); textptr^.filtype:=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(konst,'false   ',boolptr,q); enterid(p);
725       q:=p; p:=newip(konst,'true    ',boolptr,q); p^.value:=1; enterid(p);
726       boolptr^.fconst:=p;
727       p:=newip(konst,'maxint  ',intptr,nil); p^.value:=maxint; enterid(p);
728       p:=newip(konst,spaces,charptr,nil); p^.value:=maxcharord;

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

732     procedure init3;
733     var j:standpf; p:ip; q:np;
734         pfn:array[standpf] of alpha;
735         ftype:array[feof..farctan] of sp;
736     begin
737     {names of standard procedures/functions}
738       pfn[pread    ]:='read    ';  pfn[preadln  ]:='readln  ';
739       pfn[pwrite   ]:='write   ';  pfn[pwriteln ]:='writeln ';
740       pfn[pput     ]:='put     ';  pfn[pget     ]:='get     ';
741       pfn[ppage    ]:='page    ';  pfn[preset   ]:='reset   ';
742       pfn[prewrite ]:='rewrite ';  pfn[pnew     ]:='new     ';
743       pfn[pdispose ]:='dispose ';  pfn[ppack    ]:='pack    ';
744       pfn[punpack  ]:='unpack  ';  pfn[pmark    ]:='mark    ';
745       pfn[prelease ]:='release ';  pfn[phalt    ]:='halt    ';
746       pfn[feof     ]:='eof     ';  pfn[feoln     ]:='eoln    ';
747       pfn[fabs     ]:='abs     ';  pfn[fsqr      ]:='sqr     ';
748       pfn[ford     ]:='ord     ';  pfn[fchr      ]:='chr     ';
749       pfn[fpred    ]:='pred    ';  pfn[fsucc     ]:='succ    ';
750       pfn[fodd     ]:='odd     ';  pfn[ftrunc    ]:='trunc   ';
751       pfn[fround   ]:='round   ';  pfn[fsin      ]:='sin     ';
752       pfn[fcos     ]:='cos     ';  pfn[fexp      ]:='exp     ';
753       pfn[fsqrt    ]:='sqrt    ';  pfn[fln       ]:='ln      ';
754       pfn[farctan  ]:='arctan  ';
755     {parameter types of standard functions}
756       ftype[feof   ]:=nil;              ftype[feoln    ]:=nil;
757       ftype[fabs   ]:=nil;              ftype[fsqr     ]:=nil;
758       ftype[ford   ]:=nil;              ftype[fchr     ]:=intptr;
759       ftype[fpred  ]:=nil;              ftype[fsucc    ]:=nil;
760       ftype[fodd   ]:=intptr;           ftype[ftrunc   ]:=nil;
761       ftype[fround ]:=nil;              ftype[fsin     ]:=realptr;
762       ftype[fcos   ]:=realptr;          ftype[fexp     ]:=realptr;
763       ftype[fsqrt  ]:=realptr;          ftype[fln      ]:=realptr;
764       ftype[farctan ]:=realptr;
765     {standard procedure/function identifiers}
766       for j:=pread to phalt do
767         begin new(p,proc,standard); p^.klass:=proc;
768           p^.name:=pfn[j]; p^.pfkind:=standard; p^.key:=j; enterid(p);
769         end;
770       for j:=feof to farctan do
771         begin new(p,func,standard); p^.klass:=func; p^.idtype:=ftype[j];
772           {idtype is used not for result type but for parameter type !! }
773           p^.name:=pfn[j]; p^.pfkind:=standard; p^.key:=j; enterid(p);
774         end;
775     {program identifier}
776       progp:=newip(proc,'_main   ',nil,nil);
777     {new name space for user externals}
778       new(q,blck); q^.occur:=blck; q^.mlink:=top; q^.fname:=nil; top:=q;
779     end;

781     procedure init4;
782     var c:char;
783     begin
784     {pascal library mnemonics}
```



```
785       lmn[ELN ]:='_eln';    lmn[EFL ]:='_efl';      lmn[CLS ]:='_cls';
786       lmn[WDV ]:='_wdv';
787       lmn[OPN ]:='_opn';    lmn[GETX]:='_get';      lmn[RDI ]:='_rdi';
788       lmn[RDC ]:='_rdc';    lmn[RDR ]:='_rdr';      lmn[RDL ]:='_rdl';
789       lmn[RLN ]:='_rln';
790       lmn[CRE ]:='_cre';    lmn[PUTX]:='_put';      lmn[WRI ]:='_wri';
791       lmn[WSI ]:='_wsi';    lmn[WRC ]:='_wrc';      lmn[WSC ]:='_wso';
792       lmn[WRS ]:='_wre';    lmn[WSS ]:='_wss';      lmn[WRB ]:='_wrb';
793       lmn[WSB ]:='_wsb';    lmn[WRR ]:='_wrr';      lmn[WSR ]:='_wsr';
794       lmn[WRL ]:='_wrl';    lmn[WSL ]:='_wsl';
795       lmn[WRF ]:='_wrf';    lmn[WRX ]:='_wrx';      lmn[WSZ ]:='_wsx';
796       lmn[WLN ]:='_wln';    lmn[PAG ]:='_pag';
797       lmn[ABR ]:='_abr';    lmn[RND ]:='_rnd';      lmn[SIN ]:='_sin';
798       lmn[COS ]:='_cos';    lmn[EXPX]:='_exp';      lmn[SQT ]:='_sqt';
799       lmn[LOG ]:='_log';    lmn[ATN ]:='_atn';      lmn[ABI ]:='_abi';
800       lmn[ABL ]:='_abl';
801       lmn[BCP ]:='_bcp';    lmn[BTS ]:='_bts';      lmn[NEWX]:='_new';
802       lmn[SAV ]:='_sav';    lmn[RST ]:='_rst';      lmn[INI ]:='_ini';
803       lmn[HLT ]:='_hlt';    lmn[ASS ]:='_ass';      lmn[GTO ]:='_gto';
804       lmn[PAC ]:='_pac';    lmn[UNP ]:='_unp';      lmn[DIS ]:='_dis';
805       lmn[ASZ ]:='_asz';    lmn[NDX ]:='_ndi';      lmn[MDL ]:='_mdl';
806     {options}
807       for c:='a' to 'z' do begin opt[c]:=0; forceopt[c]:=false end;
808       opt['s']:=on;
809       opt['f']:=floatsize div wordsize;     {default real size in words}
810       opt['i']:=maxsetlist+1;
811       opt['l']:=on;
812       opt['o']:=on;
813       opt['p']:=addrsize div wordsize;      {default pointer size in words}
814       opt['r']:=on;
815       dopt:=off;
816     {smaller variables}
817       b.nesttop:=nil;
818       b.le:=0;
819       b.illmo:=0;
820       b.forwcount:=0;
821       b.lchain:=nil;
822       c.ohno:=0;
823       c.line:=1;
824       c.linr:=1;
825       c.orig:=1;
826       c.fname:=emptyfnam;
827       source:=emptyfnam;
828       line:=0;
829       dlmo:=0;
830       srge:=1;
831       lastpfno:=0;
832       givealine:=true;
833       isecluding:=false;
834       confexpected:=false;
835       intypedcase:=false;
836       fixmsed:=false;
837       secondtok:=false;
838       iop[false]:=nil;
839       iop[true]:=nil;
840       argv[0],ad:=-1;

841       argv[1],ad:=-1;
842     end;

844     procedure handleopts;
845     begin
846       oopt:=opt['o'];
847       dopt:=opt['d'];
848       iopt:=opt['i'];
849       sopt:=opt['s'];
850       realsize:=sopt['f'] * wordsize; realptr^.size:=realsize;
851       ptrsize:=sopt['p'] * wordsize; nilptr^.size:=ptrsize;
852       fhsize:=6*intsize + 2*ptrsize;
853       textptr^.size:=fhsize+bufsize; stringptr^.size:=ptrsize;
854       if sopt<>off then begin oopt:=off; dopt:=off end
855       else if opt['u']<>off then os['_']:=lower;
856       if oopt<>off then enterid(newip(types,'string  ',stringptr,nil));
857       if dopt<>off then enterid(newip(types,'long    ',longptr,nil));
858       if opt['o']=off then begin gen1(ps_mes,mesoptoff); genend end;
859       if ptrsize<>wordsize then begin gen1(ps_mes,mesvirtual); genend end;
860       if dopt<>off then fltused:=true;  {temporary kludge}
861     end;

863     {=============================================================}

865     procedure trace(tname:alpha; fip:ip; var namdlb:integer);
866     var i:integer;
867     begin
868       if opt['t']<>off then
869         begin
870           if namdlb=0 then
871             begin dlbno:=dlbno+1; namdlb:=dlbno; gendlb(dlbno);
872               gen0(ps_run); write(eml,sp_scon,8);
873               for i:=1 to 8 do write(eml,ord(fip^.name[i]));  genend;
874             end;
875           gen1(op_ujp,9); gen0(op_lao,namdlb);
876           gen0(op_cal); gen1dnt(sp_pnam,tname);
877         end;
878     end;

880     function forwof(fsp:sp; forus:forwset):boolean;
881     begin if fsp=nil then forwof:=false else forwof:=fsp^.form in forus end;

883     function sizeof(fsp:sp):integer;
884     var s:integer;
885     begin s:=0;
886       if fsp<>nil then s:=fsp^.size;
887       if s<>1 then if odd(s) then s:=s+1;
888       sizeof:=s
889     end;

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

894     procedure exchange(l1,l2:integer);
895     var d1,d2:integer;
896     begin d1:=l2-l1; d2:=limo-l2;
```

```
897       if (d1<>0) and (d2<>0) then
898         begin gen1(pa_exc,d1); genost(d2) end
899     end;

901     procedure setop(m:byte);
902     begin gen1(m,even(sizeof(s.asp))) end;

904     procedure expandemptyset(fsp:sp);
905     var i:integer;
906     begin
907       for i:=2 to sizeof(fsp) div wordsize do gen1(op_loc,0); s.asp:=fsp
908     end;

910     procedure push(local:boolean; sd:integer; sz:integer);
911     begin assert not odd(sz);
912       if sz>wordsize then
913         begin if local then gen1(op_lal,sd) else gen1(op_lae,sd);
914           gen1(op_loi,sz)
915         end
916       else
917         if local then gen1(op_lol,sd) else gen1(op_loe,sd)
918     end;

920     procedure pop(local:boolean; sd:integer; sz:integer);
921     begin assert not odd(sz);
922       if sz>wordsize then
923         begin if local then gen1(op_lal,sd) else gen1(op_lae,sd);
924           gen1(op_sti,sz)
925         end
926       else
927         if local then gen1(op_stl,sd) else gen1(op_ste,sd)
928     end;

930     procedure lexical(m:byte; lv:integer; sd:integer; sz:integer);
931     begin gen1(op_lex,level-lv); gen1(op_adi,sd); gen1(m,sz) end;

933     procedure loadpos(var p:position; sz:integer);
934     begin with p do
935       if lv<>0 then
936     #ifdef SEGMENTS
937         if sg<>0 then
938           begin gen1(op_lsa,sg); gen1(op_adi,sd); gen1(op_loi,sz) end
939         else
940     #endif
941           push(global,sd,sz)
942       else
943         if lv=level then push(local,sd,sz) else
944           lexical(op_loi,lv,sd,sz);
945     end;

947     procedure descraddr(var p:position);
948     begin if p.lv=0 then gen0(op_lae,p.sd) else loadpos(p,ptrsize) end;

950     procedure loadaddr;
951     begin with s do begin
952       case ak of
953         fixed:
954           with pos do
955             if lv<>0 then
956     #ifdef SEGMENTS
957               if sg<>0 then
958                 begin gen1(op_lsa,sg); gen1(op_adi,sd) end
959               else
960     #endif
961                 gen1(op_lae,sd)
962             else
963               if lv=level then gen1(op_lal,sd) else
964                 begin gen1(op_lex,level-lv); gen1(op_adi,sd) end;
965         pfixed:
966           loadpos(pos,ptrsize);
967         ploaded:
968           ;
969         indexed:
970           gen0(op_sas);
971       end;  {case}
972       ak:=ploaded;
973     end end;

975     procedure load;
976     var sz:integer;
977     begin with s do begin
978       sz:=sizeof(asp); if not packbit then sz:=even(sz);
979       if asp<>nil then
980         case ak of
981           cst:
982             gen1(op_loc,pos.sd);  {only one-word scalars}
983           fixed:
984             loadpos(pos,sz);
985           pfixed:
986             begin loadpos(pos,ptrsize); gen1(op_loi,sz) end;
987           loaded:
988             ;
989           ploaded:
990             gen1(op_loi,sz);
991           indexed:
992             gen0(op_las);
993         end;  {case}
994       ak:=loaded;
995     end end;

997     procedure store;
998     var sz:integer;
999     begin with s do begin
1000      sz:=sizeof(asp); if not packbit then sz:=even(sz);
1001      if asp<>nil then
1002        case ak of
1003          fixed:
1004            with pos do
1005              if lv<>0 then
1006    #ifdef SEGMENTS
1007                if sg<>0 then
1008                  begin gen1(op_lsa,sg);
```



```
1009                gen1(op_adi,sd); gen1(op_sti,sz)
1010              end
1011            else
1012    #endif
1013              pop(global,sd,sz)
1014          else
1015            if level=lv then pop(local,sd,sz) else
1016              lexical(op_sti,lv,sd,sz);
1017        pfixed:
1018          begin loadpos(pos,ptrsize); gen1(op_sti,sz) end;
1019        ploaded:
1020          gen1(op_sti,sz);
1021        indexed:
1022          gen0(op_sas);
1023      end;  {case}
1024    end end;

1026    procedure fieldaddr(off:integer);
1027    begin with s do
1028      if (ak=fixed) and not packbit then pos.sd:=pos.sd+off else
1029        begin loadaddr; gen1(op_adi,off) end
1030    end;

1032    procedure loadcheap;
1033    begin if formof(s.asp,[arrays..records]) then loadaddr else load end;

1035    {*****************************************************************}

1037    procedure nextch;
1038    begin
1039      eol:=eoln(input); read(input,ch); s.chno:=s.chno+1; chsy:=sss[ch];
1040    end;

1042    procedure nextln;
1043    begin
1044      if eof(input) then
1045        begin
1046          if not eofexpected then error(403) else
1047          begin
1048            if fltused then begin gen1(ps_mes,mesfloats); genend end;
1049            gen0(ps_eof)
1050          end;
1051    #ifdef STANDARD
1052          goto 9999
1053    #endif
1054    #ifndef STANDARD
1055          halt
1056    #endif
1057        end;
1058      s.chno:=0; s.lino:=s.lino+1; s.liar:=s.liar+1;
1059      if not ineluding then
1060        begin s.orig:=s.orig+1; givaline:=true end;
1061    end;

1063    procedure options(normal:boolean);
1064    var o,ci:char; i:integer;

1066    procedure getc;
1067    var b:byte;
1068    begin
1069      if normal then
1070        begin nextch; o:=ch end
1071      else
1072        begin read(em1,b); o:=chr(b) end
1073    end;

1075    begin
1076      repeat getc;
1077        if (o>='a') and (o<='z') then
1078          begin ci:=o; getc; i:=0;
1079            if o='+' then begin i:=1; getc end else
1080            if o='-' then getc else
1081            if o in[o]+digit then
1082              repeat i:=i*10 + ord(o) - ord('0'); getc;
1083              until o in[o]<>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 o<>',';
1092    end;

1094    procedure linedirective;
1095    var i,j:integer;
1096    begin i:=0; j:=0;
1097      repeat nextch until (ch<>' ') or eol;
1098      while chsy=digit do
1099        begin i:=i*10 + ord(ch) - ord('0'); nextch end;
1100      while (ch=' ') and not eol do nextch;
1101      if (ch<>'"') or (i=0) then error(404) else
1102        begin nextch;
1103          while (ch<>'"') and not eol do
1104            begin
1105              if ch='/' then j:=0 else
1106                begin if j<0 then s.fnam:=emptyfnam;
1107                  j:=j+1; if j<=fnmax then s.fnam[j]:=ch;
1108                end;
1109              nextch
1110            end;
1111          if source=emptyfnam then source:=s.fnam;
1112          inoluding:=source<>s.fnam;
1113          i:=i-1; s.liar:=i;
1114          if not inoluding then s.orig:=i
1115        end;
1116      while not eol do nextch;
1117    end;

1119    procedure putdig;
1120    begin ix:=ix+1; if ix<=rmax then strbuf[ix]:=ch; nextch end;
```

```
1122    procedure inident;
1123    label 1;
1124    var i,k:integer;
1125    begin k:=0; id:=spaces;
1126      repeat
1127        if ch=>upper then ch:=chr(ord(ch)-ord('A')+ord('a'));
1128        if k<idmax then begin k:=k+1; id[k]:=ch end;
1129        nextch
1130      until ch=>digit;
1131                 {lower=0,upper=1,digit=2. ugly but fast}
1132      for i:=frw[k-1] to frw[k] - 1 do
1133        if rw[i]=id then
1134          begin sy:=rsy[i]; goto 1 end;
1135      sy:=ident;
1136    1:
1137    end;

1139    procedure innumber;
1140    label   1;
1141    const   imax = 10;
1142    var     i:integer;
1143            is:packed array[1..imax] of char;
1144    begin ix:=0; sy:=intcst; 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 secondd0t:=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='+') or (ch='-') then putdig;
1158              if ch=>digit then error(+06) else
1159                repeat putdig until ch=>digit;
1160            end;
1161          if ix>rmax then begin error(+07); ix:=rmax end;
1162          sy:=realcst; fltused:=true; dlbno:=dlbno+1; val:=dlbno;
1163          gendlb(dlbno); gen0(pa_rom); write(em1,sp_room,ix);
1164          for i:=1 to ix do write(em1,ord(strbuf[i])); genend;
1165        end;
1166    1:if (ch='a')or(ch='b') then inident else
1167      if sy=intcst then
1168        if ix>imax then error(+08) else
1169          begin ix:='0000000000'; i:=imax+1;
1170            while ix>0 do
1171              begin i:=i-1; is[i]:=strbuf[ix]; ix:=ix-1 end;
1172            if ix<=maxintstring then
1173              while i<=imax do
1174                begin val:=val*10 - ord('0') + ord(is[i]); i:=i+1 end
1175            else if (ix<=maxlongstring) and (dopt<>off) then
1176              begin sy:=longcst; dlbno:=dlbno+1; val:=dlbno;
1177                gendlb(dlbno); gen0(pa_con); write(em1,sp_loon,imax+1-i);
1178                while i<=imax do
1179                  begin write(em1,ord(is[i])); i:=i+1 end;
1180                genend
1181              end
1182            else error(+09)
1183          end
1184    end;

1186    procedure instring(qc:char);
1187    var i:integer;
1188    begin ix:=0; zerostring:=qc='';
1189      repeat
1190        repeat nextch; ix:=ix+1; if ix<=smax then strbuf[ix]:=ch;
1191          until (ch=qc) or eol;
1192        if ch=qc then nextch else error(+010);
1193      until ch<>qc;
1194      if not zerostring then
1195        begin ix:=ix-1; if ix=0 then error(+011) end
1196      else
1197        begin strbuf[ix]:=chr(0); if opt=off then error(+012) end;
1198      if (ix=1) and not zerostring then
1199        begin sy:=charcst; val:=ord(strbuf[1]) end
1200      else
1201        begin sy:=stringcst; dlbno:=dlbno+1; val:=dlbno;
1202          if ix>smax then begin error(+013); ix:=smax end;
1203          gendlb(dlbno); gen0(pa_rom); write(em1,sp_soon,ix);
1204          for i:=1 to ix do write(em1,ord(strbuf[i])); genend;
1205        end
1206    end;

1208    procedure incomment;
1209    var stopc:char;
1210    begin nextch; stopc:=':';
1211      if ch='$' then options(true);
1212      while (ch<>'}') and (ch<>stopc) do
1213        begin stopc:='}'; if ch='*' then stopc:=':';
1214          if ch=';' then error(-(+018));
1215          if eol then nextln; nextch
1216        end;
1217      if ch<>'}' then teststandard;
1218      nextch
1219    end;

1221    procedure insym;
1222      {read next basic symbol of source program and return its
1223      description in the global variables sy, op, id, val and ix}
1224    label 1;
1225    begin
1226    1:case chsy of
1227        tabch:
1228          begin e.chno:=e.chno - e.chno mod 8 + 8; nextch; goto 1 end;
1229        layout:
1230          begin if eol then nextln; nextch; goto 1 end;
1231        lower,upper: inident;
1232        digit: innumber;
```



```
1233        quotech,dquotech:
1234          instring(ch);
1235        colonch:
1236          begin nextch;
1237            if ch='=' then begin sy:=becomes; nextch end else sy:=colon;
1238            end;
1239        periodch:
1240          begin nextch;
1241            if secondd0t then begin secondd0t:=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:=leqy; nextch end else
1247            if ch='>' then begin sy:=neqy; nextch end else sy:=ltgy
1248            end;
1249        greaterch:
1250          begin nextch;
1251            if ch='=' then begin sy:=geqy; nextch end else sy:=gtgy
1252            end;
1253        lparentch:
1254          begin nextch;
1255            if ch<>'*' then sy:=lparent else
1256              begin teststandard; incomment; goto 1 end;
1257            end;
1258        lbracech:
1259          begin incomment; goto 1 end;
1260        rparentch,lbrackch,rbrackch,commach,semich,arrowch,
1261        plusch,minch,slash,star,equal:
1262          begin sy:=sxy[chsy]; nextch end;
1263        others:
1264          begin
1265            if (ch='$') and (e.chno=1) then linedirective else
1266              begin error(+015); nextch end;
1267            goto 1
1268          end;
1269      end {case}
1270    end;

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

1275    function find1(sys1,sys2:sos; err:integer):boolean;
1276    {symbol of sys1 expected. return true if sy in sys1}
1277    begin
1278      if not (sy in sys1) then
1279        begin error(err); while not (sy in sys1+sys2) do insym end;
1280      find1:=sy in sys1
1281    end;

1283    function find2(sys1,sys2:sos; err:integer):boolean;
1284    {symbol of sys1+sys2 expected. return true if sy in sys1}
1285    begin
1286      if not (sy in sys1+sys2) then
1287        begin error(err); repeat insym until sy in sys1+sys2 end;
1288      find2:=sy in sys1
```

```
1289    end;

1291    function find3(sy1:symbol; sys2:sos; err:integer):boolean;
1292    {symbol sy1 or one of sys2 expected. return true if sy1 found and skip}
1293    begin find3:=true;
1294      if not (sy in [sy1]+sys2) then
1295        begin error(err); repeat insym until sy in [sy1]+sys2 end;
1296      if sy=sy1 then insym else find3:=false
1297    end;

1299    function endofloop(sys1,sys2:sos; sy3:symbol; err:integer):boolean;
1300    begin endofloop:=false;
1301      if find2(sys2+[sy3],sys1,err) then nextif(sy3,err+1)
1302      else endofloop:=true;
1303    end;

1305    function lastsemicolon(sys1,sys2:sos; err:integer):boolean;
1306    begin lastsemicolon:=true;
1307      if not endofloop(sys1,sys2,semicolon,err) then
1308        if find2(sys2,sys1,err+2) then lastsemicolon:=false
1309    end;

1311    {*************************************************************}

1313    function searchid(fids1a: setofids):ip;
1314    {search for current identifier symbol in the name table}
1315    label 1;
1316    var lip:ip; ic:idclass;
1317    begin lastsp:=top;
1318      while lastsp<>nil do
1319        begin lip:=lastsp^.fname;
1320          while lip<>nil do
1321            if lip^.name=id then
1322              if lip^.klass in fids1 then
1323                begin
1324                  if lip^.klass=vars then if lip^.vpos.lv<>level then
1325                    lip^.iflag:=lip^.iflag+[moreq];
1326                  goto 1
1327                end
1328              else lip:=lip^.rlink
1329            else
1330              if lip^.name< id then lip:=lip^.rlink else lip:=lip^.llink;
1331          lastsp:=lastsp^.slink;
1332        end;
1333      errid(+016,id);
1334      if types in fids1 then ic:=types else
1335      if vars  in fids1 then ic:=vars else
1336      if hconst in fids1 then ic:=hconst else
1337      if proc  in fids1 then ic:=proc else
1338      if func  in fids1 then ic:=func else ic:=field;
1339      lip:=undefip[ic];
1340    1:
1341      searchid:=lip
1342    end;

1344    function searchsection(fip: ip):ip;
```

```
1345    {to find record fields and forward declared procedure id's
1346      -->procedure pfdeclaration
1347      -->procedure selector}
1348    label 1;
1349    begin
1350      while fip<>nil do
1351        if fip^.name=id then goto 1 else
1352          if fip^.name< id then fip:=fip^.rlink else fip:=fip^.llink;
1353    1:  searchsection:=fip
1354    end;

1356    function searchlab(flp:lp; val:integer):lp;
1357    label 1;
1358    begin
1359      while flp<>nil do
1360        if flp^.labval=val then goto 1 else flp:=flp^.nextlp;
1361    1:searchlab:=flp
1362    end;

1364    procedure opconvert(ts:twostruct);
1365    var op:integer;
1366    begin with s do begin
1367      case ts of
1368        ir: begin op:=op_oif; asp:=realptr; fltused:=true end;
1369        ri: begin op:=op_ofi; asp:=intptr; fltused:=true end;
1370        il: begin op:=op_oid; asp:=longptr end;
1371        li: begin op:=op_odi; asp:=intptr end;
1372        lr: begin op:=op_odf; asp:=realptr; fltused:=true end;
1373        rl: begin op:=op_ofd; asp:=longptr; fltused:=true end;
1374      end;
1375      gen0(op)
1376    end end;

1378    procedure negate(l1:integer);
1379    var l2:integer;
1380    begin
1381      if s.asp=intptr then gen0(op_neg) else
1382        begin l2:=line; gen1(op_loc,0);
1383          if s.asp=longptr then
1384            begin opconvert(il); exchange(l1,l2); gen0(op_dab) end
1385          else  {realptr}
1386            begin opconvert(ir); exchange(l1,l2); gen0(op_fsb) end
1387        end
1388    end;

1390    function desub(fsp:sp):sp;
1391    begin
1392      if formof(fsp,[subrange]) then fsp:=fsp^.rangetype; desub:=fsp
1393    end;

1395    function nicescalar(fsp:sp):boolean;
1396    begin
1397      if fsp=nil then nicescalar:=true else
1398        nicescalar:=(fsp^.form=scalar) and (fsp<>realptr) and (fsp<>longptr)
1399    end;

1401    function bounds(fsp:sp; var fmin,fmax:integer):boolean;
1402    {compute bounds if possible, else return false}
1403    begin bounds:=false; fmin:=0; fmax:=0;
1404      if fsp<>nil then
1405        if fsp^.form=subrange then
1406          begin fmin:=fsp^.min; fmax:=fsp^.max; bounds:=true end else
1407        if fsp^.form=scalar then
1408          if fsp^.fconst<>nil then
1409            begin fmin:=0; fmax:=fsp^.fconst^.value; bounds:=true end
1410    end;

1412    procedure genrok(fsp:sp);
1413    var min,max,sno:integer;
1414    begin
1415      if opt['r']<>off then if bounds(fsp,min,max) then
1416        begin
1417          if fsp^.form=scalar then sno:=fsp^.scalno else sno:=fsp^.subrno;
1418          if sno=0 then
1419            begin dlbno:=dlbno+1; sno:=dlbno;
1420              gendlb(dlbno); gen1(ps_rom,min); genost(max); genend;
1421              if fsp^.form=scalar then fsp^.scalno:=sno else
1422                fsp^.subrno:=sno
1423            end;
1424          gend(op_rok,sno);
1425        end
1426    end;

1428    procedure checkbnds(fsp:sp);
1429    var min1,max1,min2,max2:integer; bool:boolean;
1430    begin
1431      if bounds(fsp,min1,max1) then
1432        begin bool:=bounds(s.asp,min2,max2);
1433          if (bool=false) or (min2<min1) or (max2>max1) then
1434            genrok(fsp);
1435        end;
1436      s.asp:=fsp;
1437    end;

1439    function eqstruct(p,q:sp):boolean;
1440    begin eqstruct:=(p=q) or (p=nil) or (q=nil) end;

1442    function string(fsp:sp):boolean;
1443    var lsp:sp;
1444    begin string:=false;
1445      if formof(fsp,[arrays]) then
1446        if eqstruct(fsp^.aeltype,charptr) then
1447          if spack in fsp^.sflag then
1448            begin lsp:=fsp^.inxtype;
1449              if lsp=nil then string:=true else
1450                if lsp^.form=subrange then
1451                  if lsp^.rangetype=intptr then
1452                    if lsp^.min=1 then
1453                      string:=true
1454            end
1455    end;
```



```
1457    function compat(p,q:sp):twostruct;
1458    begin compat:=noteq;
1459      if eqstruct(p,q) then compat:=eq else
1460        begin p:=desub(p); q:=desub(q);
1461          if eqstruct(p,q) then compat:=subeq else
1462          if p^.form=q^.form then
1463            case p^.form of
1464              scalar:
1465                if (p=intptr) 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=nilptr) or (q=nilptr) then compat:=eq;
1474              power:
1475                if p=emptyset then compat:=es else
1476                if q=emptyset then compat:=se else
1477                if ocompat(p^.elset,q^.elset) <= subeq then
1478                  if p^.sflag=q^.sflag then compat:=eq;
1479              arrays:
1480                if string(p) and string(q) and (p^.size=q^.size) then
1481                  compat:=eq;
1482              files,carray,records: ;
1483            end;
1484        end
1485    end;

1487    procedure checkasp(fsp:sp; err:integer);
1488    var ts:twostruct;
1489    begin
1490      ts:=compat(s.asp,fsp);
1491      case ts of
1492        eq:
1493          if fsp<>nil then if withfile in fsp^.sflag then asperr(err);
1494        subeq:
1495          checkbnds(fsp);
1496        li:
1497          begin opconvert(ts); checkasp(fsp,err) end;
1498        il,rl,lr,ir:
1499          opconvert(ts);
1500        es:
1501          expandemptyset(fsp);
1502        noteq,ri,se:
1503          asperr(err);
1504      end
1505    end;

1507    procedure force(fsp:sp; err:integer);
1508    begin load; checkasp(fsp,err) end;

1510    function newident(kl:idclass; idt:sp; nxt:ip; err:integer):ip;
1511    begin newident:=nil;
1512      if sy<>ident then error(err) else
1513        begin newident:=newip(kl,id,idt,nxt); insym end
1514    end;

1516    function stringstruct:sp;
1517    var lsp:sp;
1518    begin {only used when ix and zerostring are still valid}
1519      if zerostring then lsp:=stringptr else
1520        begin lsp:=newsp(arrays,ix*charsize); lsp^.sflag:=[spack];
1521          lsp^.aeltype:=charptr; lsp^.inxtype:=nil;
1522        end;
1523      stringstruct:=lsp;
1524    end;

1526    function address(var lc:integer; sz:integer; pack:boolean):integer;
1527    begin
1528      if lc >= maxint-sz then begin error(+017); lc:=0 end;
1529      if (not pack) or (sz>1) then if odd(lc) then lc:=lc+1;
1530      address:=lc;
1531      lc:=lc+sz
1532    end;

1534    function reserve(s:integer):integer;
1535    var r:integer;
1536    begin r:=address(b.lc,s,false); genreg(r,s,100); reserve:=r;
1537      if b.lc>lcmax then lcmax:=b.lc
1538    end;

1540    function arraysize(fsp:sp; pack:boolean):integer;
1541    var es,min,max,tot,n:integer;
1542    begin es:=sizeof(fsp^.aeltype);
1543      if not pack then es:=wrem(es);
1544      if bounds(fsp^.inxtype,min,max) then;  {we checked before}
1545      dlbno:=dlbno+1; fsp^.arpos.lv:=0; fsp^.arpos.ad:=dlbno;
1546      gendlb(dlbno); gen1(ps_rom,min); genost(max-min);
1547      genost(es); genend;
1548      n:=max-min+1; tot:=es*n;
1549      if es<>0 then if tot div es <> n then begin error(+018); tot:=0 end;
1550      arraysize:=tot
1551    end;

1553    procedure treewalk(fip:ip);
1554    var lsp:sp; i:integer;
1555    begin
1556      if fip<>nil then
1557        begin treewalk(fip^.llink); treewalk(fip^.rlink);
1558          if fip^.klass=vars then
1559            begin if not (used in fip^.iflag) then errid(-(+019),fip^.name);
1560              if not (assigned in fip^.iflag) then errid(-(+020),fip^.name);
1561              lsp:=fip^.idtype;
1562              if not (noreg in fip^.iflag) then
1563                genreg(fip^.vpos.ad,sizeof(lsp),ord(formof(lsp,[pointer])));
1564              if lsp<>nil then if withfile in lsp^.sflag then
1565                if lsp^.form=files then
1566                  if level=1 then
1567                    begin
1568                      for i:=2 to argc do with argv[i] do
```

```
1569                    if name=fip^.name then ad:=fip^.vpos.ad
1570                  end
1571                else
1572                  begin
1573                    if not (refer in fip^.iflag) then
1574                      begin gen1(op_mrk,0);
1575                        gen1(op_lal,fip^.vpos.ad); genxp(CLS)
1576                      end
1577                  end
1578              else
1579                if level<>1 then errid(-(+021),fip^.name)
1580            end
1581        end
1582    end;

1584    procedure constant(fsys:sos; var fsp:sp; var fval:integer);
1585    var signed,min:boolean; lip:ip;
1586    begin signed:=(sy=plussy) or (sy=minsy);
1587      if signed then begin min:=sy=minsy; insym end else min:=false;
1588      if find1([ident..nilcst],fsys,+022) then
1589        begin fval:=val;
1590          case sy of
1591            stringcst: fsp:=stringstruct;
1592            charcst: fsp:=charptr;
1593            intcst: fsp:=intptr;
1594            realcst: fsp:=realptr;
1595            longcst: fsp:=longptr;
1596            nilcst: fsp:=nilptr;
1597            ident:
1598              begin lip:=searchid([konst]);
1599                fsp:=lip^.idtype; fval:=lip^.value;
1600              end
1601          end;  {case}
1602          if signed then
1603            if (fsp<>intptr) and (fsp<>realptr) and (fsp<>longptr) then
1604              error(+023)
1605            else if min then fval:= -fval;
1606                      {note: negating the v-number for reals and longs}
1607          insym;
1608        end
1609      else begin fsp:=nil; fval:=0 end;
1610    end;

1612    function cstinteger(fsys:sos; fsp:sp; err:integer):integer;
1613    var lsp:sp; lval,min,max:integer;
1614    begin constant(fsys,lsp,lval);
1615      if fsp<>lsp then
1616        if comptypes(desub(fsp),lsp) then
1617          begin
1618            if boundns(fsp,min,max) then
1619              if (lval<min) or (lval>max) then error(+024)
1620          end
1621        else
1622          begin error(err); lval:=0 end;
1623      cstinteger:=lval
1624    end;

1626    {*****************************************************************}

1628    function typid(err:integer):sp;
1629    var lip:ip; lsp:sp;
1630    begin lsp:=nil;
1631      if sy<>ident then error(err) else
1632        begin lip:=searchid([types]); lsp:=lip^.idtype; insym end;
1633      typid:=lsp
1634    end;

1636    function simpletyp(fsys:sos):sp;
1637    var lsp,lsp1:sp; lip,hip:ip; min,max:integer; lnp:np;
1638        newsubrange:boolean;
1639    begin lsp:=nil;
1640      if find1([ident..lparent],fsys,+025) then
1641        if sy=lparent then
1642          begin insym; lnp:=top;   {decl. consts local to innermost block}
1643            while top^.occur<>blck do top:=top^.slink;
1644            lsp:=newsp(scalar,wordsize); hip:=nil; max:=0;
1645            repeat lip:=newident(konst,lsp,hip,+026);
1646              if lip<>nil then
1647                begin enterid(lip);
1648                  hip:=lip; lip^.value:=max; max:=max+1
1649                end;
1650            until endofloop(fsys+[rparent],[ident],comma,+027);  {+028}
1651            if max<t8 then lsp^.size:=bytesize;
1652            lsp^.fconst:=hip; top:=lnp; nextif(rparent,+029);
1653          end
1654        else
1655          begin newsubrange:=true;
1656            if sy=ident then
1657              begin lip:=searchid([types,konst]); insym;
1658                if lip^.klass=types then
1659                  begin lsp:=lip^.idtype; newsubrange:=false end
1660                else
1661                  begin lsp1:=lip^.idtype; min:=lip^.value end
1662              end
1663            else constant(fsys+[colon2,ident..plussy],lsp1,min);
1664            if newsubrange then
1665              begin lsp:=newsp(subrange,wordsize); lsp^.subrno:=0;
1666                if not scalar(lsp1) then
1667                  begin error(+030); lsp1:=nil; min:=0 end;
1668                lsp^.rangetype:=lsp1;
1669                nextif(colon2,+031); max:=cstinteger(fsys,lsp1,+032);
1670                if min>max then begin error(+033); max:=min end;
1671                if (min>=0) and (max<t8) then lsp^.size:=bytesize;
1672                lsp^.min:=min; lsp^.max:=max
1673              end
1674          end;
1675      simpletyp:=lsp
1676    end;

1678    function arraytyp(fsys:sos;
1679                      artyp:structform;
1680                      aflag:aflagset;
```

```
1681                    function element(fsys:sos):sp
1682                    ):sp;
1683    var lsp,lsp1,hsp:sp; min,max:integer; ok:boolean; sepsy:symbol; lip:ip;
1684        oksys:sos;
1685    begin insym; nextif(lbrack,+034); hsp:=nil;
1686      repeat lsp:=newsp(artyp,0); initpos(lsp^.apos);
1687        lsp^.eltype:=hsp; hsp:=lsp;  {link reversed}
1688        if artyp=carray then
1689          begin sepsy:=semicolon; oksys:=[ident];
1690            lip:=newident(varrbnd,lsp,nil,+035);
1691            if lip<>nil then enterid(lip);
1692            nextif(colon2,+036);
1693            lip:=newident(varrbnd,lsp,lip,+037);
1694            if lip<>nil then enterid(lip);
1695            nextif(colon1,+038); lsp1:=typid(+039);
1696            ok:=micosscalar(desub(lsp1));
1697          end
1698        else
1699          begin sepsy:=comma; oksys:=[ident..lparent];
1700            lsp1:=simpletyp(fsys+[comma,rbrack,ofsy,ident..packedsy]);
1701            ok:=boundns(lsp1,min,max)
1702          end;
1703        if not ok then begin error(+040); lsp1:=nil end;
1704        lsp^.inxtype:=lsp1
1705      until endofloop(fsys+[rbrack,ofsy,ident..packedsy],oksys,
1706                      sepsy,+041);  {+042}
1707      nextif(rbrack,+043); nextif(ofsy,+044);
1708      lsp:=element(fsys);
1709      if lsp<>nil then aflag:=aflag + lsp^.aflag * [withfile];
1710      repeat  {reverse links and compute size}
1711        hsp:=lsp^.eltype; lsp^.eltype:=hsp; hsp^.aflag:=aflag;
1712        if artyp=arrays then lsp^.size:=arraysize(hsp,speck in aflag);
1713        lsp:=hsp; hsp:=lsp1
1714      until hsp=nil;  {lsp points to array with highest dimension}
1715      arraytyp:=lsp
1716    end;

1718    function typ(fsys:sos):sp;
1719    var lsp,lsp1:sp; oc,sz,min,max:integer;
1720        aflag:aflagset; lnp:np;

1722    function fldlist(fsys:sos):sp;
1723          {level 2: <<  typ}
1724    var fip,hip,lip:ip; lsp:sp;

1726    function varpart(fsys:sos):sp;
1727          {level 3: <<  fldlist <<  typ}
1728    var tip,lip:ip; lvp,headsp,hsp,vsp,tsp,tsp1,tfsp:sp;
1729        minoc,maxoc,int,uvar:integer; lid:alpha;
1730    begin insym; tip:=nil; lip:=nil;
1731      tsp:=newsp(tag,0);
1732      if sy<>ident then error(+045) else
1733        begin lid:=id; insym;
1734          if sy=colon1 then
1735            begin tip:=enterip(field,lid,nil,nil); enterid(tip); insym;
1736              if sy<>ident then error(+046) else
```

```
1737                begin lid:=id; insym end;
1738            end;
1739          if sy=ofsy then  {otherwise you may destroy id}
1740            begin id:=lid; lip:=searchid([types]) end;
1741        end;
1742      if lip=nil then tfsp:=nil else tfsp:=lip^.idtype;
1743      if bounds(tfsp,int,uvar) then uvar:=uvar-int+1 else
1744        begin uvar:=0;
1745          if tfsp<>nil then begin error(+047); tfsp:=nil end
1746        end;
1747      tsp^.tfldsp:=tfsp;
1748      if tip<>nil then  {explicit tag}
1749        begin tip^.idtype:=tfsp;
1750          tip^.foffset:=address(oc,sizeof(tfsp),speck in aflag)
1751        end;
1752      nextif(ofsy,+048); minoc:=oc; maxoc:=minoc; headsp:=nil;
1753      repeat hsp:=nil;  {for each caselabel list}
1754        repeat uvar:=uvar-1;
1755          int:=cstinteger(fsys+[ident..plussy,comma,colon1,lparent,
1756                                semicolon,casesy,rparent],tfsp,+049);
1757          lsp:=headsp;  {each label may occur only once}
1758          while lsp<>nil do
1759            begin if lsp^.varval=int then error(+050);
1760              lsp:=lsp^.nxtvar
1761            end;
1762          vsp:=newsp(variant,0); vsp^.varval:=int;
1763          vsp^.nxtvar:=headsp; headsp:=vsp;  {chain of case labels}
1764          vsp^.subtap:=hsp; hsp:=vsp;
1765                {use this field to link labels with same variant}
1766        until endofloop(fsys+[colon1,lparent,semicolon,casesy,rparent],
1767                        [ident..plussy],comma,+051);  {+052}
1768        nextif(colon1,+053); nextif(lparent,+054);
1769        tsp1:=fldlist(fsys+[rparent,semicolon,ident..plussy]);
1770        if oc>maxoc then maxoc:=oc;
1771        while vsp<>nil do
1772          begin vsp^.size:=oc; hsp:=vsp^.subtap;
1773            vsp^.subtap:=tsp1; vsp:=hsp
1774          end;
1775        nextif(rparent,+055);
1776        oc:=minoc;
1777      until lastsemicolon(fsys,[ident..plussy],+056);  {+057 +058}
1778      if uvar>0 then error(-(+059));
1779      tsp^.fstvar:=headsp; tsp^.size:=minoc; oc:=maxoc; varpart:=tsp;
1780    end;

1782    begin  {fldlist}
1783      if find2([ident],fsys+[casesy],+060) then
1784        repeat lip:=nil; hip:=nil;
1785          repeat fip:=newident(field,nil,nil,+061);
1786            if fip<>nil then
1787              begin enterid(fip);
1788                if lip=nil then hip:=fip else lip^.next:=fip; lip:=fip;
1789              end;
1790          until endofloop(fsys+[colon1,ident..packedsy,semicolon,casesy],
1791                          [ident],comma,+062);  {+063}
1792          nextif(colon1,+064);
```

```
1793          lsp:=typ(fsys+[casesy,semicolon]);
1794          if lsp<>nil then if withfile in lsp^.sflag then
1795            sflag:=sflag+[withfile];
1796          while hip<>nil do
1797            begin hip^.idtype:=lsp;
1798              hip^.foffset:=address(cc,sizeof(lsp),spack in sflag);
1799              hip:=hip^.next
1800            end;
1801        until lastsemicolon(fsys+[casesy],[ident],+065);  {+066 +067}
1802      if sy=casesy then fldlist:=varpart(fsys) else fldlist:=nil;
1803    end;


1806    begin  {typ}
1807      sflag:=[]; lsp:=nil;
1808      if sy=packedsy then begin sflag:=[spack]; insym end;
1809      if find1([ident..filesy],fsys,+068) then
1810        if sy in [ident..arrow] then
1811          begin if spack in sflag then error(+069);
1812            if sy=arrow then
1813              begin lsp:=newsp(pointer,ptrsize); insym;
1814                if not intypedec then lsp^.eltype:=stypid(+070) else
1815                  if sy<>ident then error(+071) else
1816                    begin fwptr:=newip(types,id,lsp,fwptr); insym end
1817              end
1818            else lsp:=simpletyp(fsys);
1819          end
1820        else
1821          case sy of
1822    {<<<<<<<<<<<<}
1823    arraysy:
1824      lsp:=arraytyp(fsys,arrays,sflag,typ);
1825    recordsy:
1826      begin insym;
1827        new(lsp,rec); lsp^.occur:=rec; lsp^.mlink:=top;
1828        lsp^.fname:=nil; top:=lsp;
1829        oc:=0; lsp1:=fldlist(fsys+[endsy]);  {fldlist updates oc}
1830        lsp:=newsp(records,oc); lsp^.tagsp:=lsp1;
1831        lsp^.fstfld:=top^.fname; lsp^.sflag:=sflag;
1832        top:=top^.mlink; nextif(endsy,+072)
1833      end;
1834    setsy:
1835      begin insym; nextif(ofsy,+073); lsp1:=simpletyp(fsys);
1836        if bounds(lsp1,min,max) then lsp1:=desub(lsp1) else
1837          if lsp1=intptr then
1838            begin error(-(+074)); max:=iopt-1 end
1839          else
1840            begin error(+075); lsp1:=nil end;
1841        if lsp1=intptr then sz:=iopt-1 else
1842          begin if bounds(lsp1,min,max) then {nothing}; sz:=max end;
1843        if (min<0) or (max>sz) or (sz div bytebits >= maxsetsize) then
1844          begin error(+076); lsp1:=nil; sz:=0 end;
1845        lsp:=newsp(power,sz div bytebits +1); lsp^.elset:=lsp1;
1846      end;
1847    filesy:
1848      begin insym; nextif(ofsy,+077); lsp1:=typ(fsys);
```

```
1849        if lsp1<>nil then if withfile in lsp1^.sflag then error(-(+078));
1850        sz:=sizeof(lsp1); if sz<buffsize then sz:=buffsize;
1851        lsp:=newsp(files,sz+fhsize); lsp^.filtype:=lsp1;
1852      end;
1853    {>>>>>>>>>>>>}
1854          end;  {case}
1855      typ:=lsp;
1856    end;

1858    function vpartyp(fsys:sos):sp;
1859    begin
1860      if find2([arraysy],fsys+[ident],+079) then
1861        vpartyp:=arraytyp(fsys,carray,[],vpartyp)
1862      else
1863        vpartyp:=stypid(+080)
1864    end;

1866    {=============================================================}

1868    procedure block(fsys:sos; fip:ip); forward;
1869            {pfdeclaration calls block. With a more obscure lexical
1870              structure this forward declaration can be avoided}

1872    procedure labeldeclaration(fsys:sos);
1873    var llp:lp;
1874    begin with b do begin
1875      repeat
1876        if sy<>intcst then error(+081) else
1877          begin
1878            if searchlab(lchain,val)<>nil then errint(+082,val) else
1879              begin new(llp); llp^.labval:=val;
1880                if val>9999 then teststandard;
1881                ilbno:=ilbno+1; llp^.labname:=ilbno; llp^.labdlb:=0;
1882                llp^.seen:=false; llp^.nextlp:=lchain; lchain:=llp;
1883              end;
1884            insym
1885          end
1886      until endofloop(fsys+[semicolon],[intcst],comma,+083);  {+084}
1887      nextif(semicolon,+085)
1888    end end;

1890    procedure constdefinition(fsys:sos);
1891    var lip:ip;
1892    begin
1893      repeat lip:=newident(konst,nil,nil,+086);
1894        if lip<>nil then
1895          begin nextif(eqsy,+087);
1896            constant(fsys+[semicolon,ident],lip^.idtype,lip^.value);
1897            nextif(semicolon,+088); enterid(lip);
1898          end;
1899      until not find2([ident],fsys,+089);
1900    end;

1902    procedure typedefinition(fsys:sos);
1903    var lip:ip;
1904    begin fwptr:=nil; intypedec:=true;
```



```
1905      repeat lip:=newident(types,nil,nil,+090);
1906        if lip<>nil then
1907          begin nextif(eqsy,+091);
1908            lip^.idtype:=styp(fsys+[semicolon,ident]);
1909            nextif(semicolon,+092); enterid(lip);
1910          end;
1911      until not find2([ident],fsys,+093);
1912      while fwptr<>nil do
1913        begin assert sy<>ident;
1914          id:=fwptr^.name; lip:=searchid([types]);
1915          fwptr^.idtype^.eltype:=lip^.idtype; fwptr:=fwptr^.next
1916        end;
1917      intypedec:=false;
1918    end;

1920    procedure vardeclaration(fsys:sos);
1921    var lip,hip,vip:ip; lsp:sp;
1922    begin with b do begin
1923      repeat hip:=nil; lip:=nil;
1924        repeat vip:=newident(vars,nil,nil,+094);
1925          if vip<>nil then
1926            begin enterid(vip); vip^.iflag:=[];
1927              if lip=nil then hip:=vip else lip^.next:=vip; lip:=vip;
1928            end;
1929        until endofloop(fsys+[colon1,ident..packedsy],[ident],comma,+095);
1930                                                                {+096}
1931        nextif(colon1,+097);
1932        lsp:=typ(fsys+[semicolon,ident]);
1933        while hip<>nil do
1934          begin hip^.idtype:=lsp;
1935            hip^.vpos.ad:=address(lc,sizeof(lsp),false); hip:=hip^.next
1936          end;
1937        nextif(semicolon,+098);
1938      until not find2([ident],fsys,+099);
1939    end end;

1941    procedure pfhead(fsys:sos;
1942                     var fip:ip;
1943                     var again:boolean;
1944                     parm:boolean); forward;

1946    function parlist(fsys:sos; var klc:integer):ip;
1947    var lastip,hip,lip,pip:ip; lsp,tsp:sp; iflag:iflagset; again:boolean;
1948        sz:integer;
1949    begin parlist:=nil; lastip:=nil;
1950      repeat {comes for each formal-parameter-section}
1951        if find1([ident,varsy,procsy,funcsy],fsys+[semicolon],+0100) then
1952          begin
1953            if (sy=procsy) or (sy=funcsy) then
1954              begin
1955                pfhead(fsys+[semicolon,ident,varsy,procsy,funcsy],
1956                       hip,again,true);
1957                hip^.pfpos.ad:=address(klc,paramsize+ptrsize,false);
1958                hip^.pfkind:=formal; lip:=hip;
1959                top:=top^.mlink; level:=level-1
1960              end
```

```
1961            else
1962              begin hip:=nil; lip:=nil; iflag:=[assigned,noreg];
1963                if sy=varsy then
1964                  begin iflag:=[refer,assigned,used,noreg]; insym end;
1965                repeat pip:=newident(vars,nil,nil,+0101);
1966                  if pip<>nil then
1967                    begin enterid(pip); pip^.iflag:=iflag;
1968                      if lip=nil then hip:=pip else lip^.next:=pip;
1969                      lip:=pip;
1970                    end;
1971                  iflag:=iflag+[samesect];
1972                until endofloop(fsys+[semicolon,colon1],
1973                                [ident],comma,+0102);  {+0103}
1974                nextif(colon1,+0104);
1975                if refer in iflag then
1976                  begin lsp:=vpartyp(fsys+[semicolon]);
1977                    sz:=ptrsize; tsp:=lsp;
1978                    while formof(tsp,[carray]) do
1979                      begin tsp^.arpos.ad:=address(klc,ptrsize,false);
1980                        tsp:=tsp^.eltype
1981                      end;
1982                  end
1983                else
1984                  begin lsp:=stypid(+0105); sz:=sizeof(lsp) end;
1985                pip:=hip;
1986                while pip<>nil do
1987                  begin pip^.vpos.ad:=address(klc,sz,false);
1988                    pip^.idtype:=lsp; pip:=pip^.next
1989                  end;
1990              end;
1991            if lastip=nil then parlist:=hip else lastip^.next:=hip;
1992            lastip:=lip;
1993          end;
1994      until endofloop(fsys,[ident,varsy,procsy,funcsy],
1995                      semicolon,+0106);  {+0107}
1996    end;

1998    procedure pfhead;  {forward declared}
1999    var lip:ip; lsp:sp; lmp:mp; kl:idclass;
2000    begin lip:=nil; again:=false;
2001      if sy=procsy then kl:=proc else
2002        begin kl:=func; fsys:=fsys+[colon1,ident] end;
2003      insym;
2004      if sy<>ident then begin error(+0108); id:=spaces end;
2005      if not parm then lip:=searchsection(top^.fname);
2006      if lip<>nil then
2007        if (lip^.klass<>kl) or (lip^.pfkind<>forward) then
2008          errid(+0109,id)
2009        else
2010          begin b.forwcount:=b.forwcount-1; again:=true end;
2011      if again then insym else
2012        begin lip:=newip(kl,id,nil,nil);
2013          if sy=ident then begin enterid(lip); insym end;
2014          lastpfno:=lastpfno+1; lip^.pfno:=lastpfno;
2015        end;
2016      level:=level+1;
```