Ellis Horwood Publishers
COMPUTERS AND THEIR APPLICATIONS

Pascal Implementation

Compiler and Assembler/Interpreter

S. PEMBERTON and M.C. DANIELS

Steven Pemberton is a Lecturer in Computing at Brighton Polytechnic in Sussex. His first post was with the University of Sussex, as a Programmer in their Research Support Unit, from 1972 to 1975. He then became a Research Programmer in the Department of Computer Science, employed on a project writing a compiler for the language Algol 68 on a research computer, the MU5. He joined the Department of Computing and Cybernetics at Brighton Polytechnic as a Research Assistant in 1977, designing and implementing a new programming language, "Newspeak". He took up his present position as a Lecturer in 1980. Mr. Pemberton has much experience in computer language implementation.

Martin Daniels is Principal Lecturer in Computing at Brighton Polytechnic. He is a graduate of the University of Aston in Birmingham with a B.Sc. (First Class Hons.) in Electrical Engineering in 1969; a D.Phil. in Electronics gained from the University of Sussex in 1974, and an MIEE, C.Eng., and MBSC, in 1976. He was a Research Fellow in the University of Sussex, 1974/75; and joined Brighton Polytechnic in his present position in 1975.

Both Steven Pemberton and Martin Daniels have collaborated on a successful project to implement Pascal on an Intel 8085a microcomputer.

PASCAL IMPLEMENTATION:

Compiler and Assembler/Interpreter

STEVEN PEMBERTON and MARTIN DANIELS

Department of Computing and Cybernetics
Brighton Polytechnic

ELLIS HORWOOD LIMITED
Publishers · Chichester

Halsted Press: a division of
JOHN WILEY & SONS
New York · Brisbane · Chichester · Toronto

TABLE OF CONTENTS

The sources of the Pascal program are available in machine-readable form on magnetic tape on application to the publishers.

British Library Cataloguing in Publication Data
Pemberton, Steven
Pascal implementation. – (Ellis Horwood series in computer science)
1. PASCAL (Computer program language)
I. Title II. Daniels, Martin
001.64'24 QA76.73P2

Library of Congress Card No. 81-20184 AACR2

ISBN 0-85312-358-6 Book (Ellis Horwood Ltd., Publishers)
ISBN 0-85312-437-X Compiler (Ellis Horwood Ltd., Publishers)
ISBN 0-470-27325-9 (Halsted Press)

Typeset in Press Roman by Ellis Horwood Ltd.
Printed in Great Britain by R. J. Acford, Chichester

Compiler Listing

```
 1 (*$c+,t-,d-,l-*)
 2  (*********************************************
 3   *                                           *
 4   *                                           *
 5   *         Portable Pascal compiler          *
 6   *         ************************          *
 7   *                                           *
 8   *                Pascal P4                  *
 9   *                                           *
10   *                                           *
11   *     Authors:                              *
12   *              Urs Ammann                   *
13   *              Kesav Nori                   *
14   *              Christian Jacobi             *
15   *                                           *
16   *     Address:                              *
17   *                                           *
18   *          Institut Fuer Informatik         *
19   *          Eidg. Technische Hochschule      *
20   *          CH-8096 Zuerich                  *
21   *                                           *
22   *                                           *
23   *                                           *
24   *                                           *
25   *                                           *
26   *********************************************)
27
28
29 program pascalcompiler(input,output,prr);
30
31
32
33 const displimit = 20; maxlevel = 10;
34    intsize     =      1;
35    intal       =      1;
36    realsize    =      1;
37    realal      =      1;
38    charsize    =      1;
39    charal      =      1;
40    charmax     =      1;
41    boolsize    =      1;
42    boolal      =      1;
43    ptrsize     =      1;
44    adral       =      1;
45    setsize     =      1;
46    setal       =      1;
47    stackal     =      1;
48    stackelsize =      1;
49    strglgth    =     16;
50    sethigh     =     47;
51    setlow      =      0;
52    ordmaxchar  =     63;
53    ordminchar  =      0;
54    maxint      =  32767;
55    lcaftermarkstack = 5;
56    fileal      = charal;
```

```
    (* stackelsize = minimum size for 1 stackelement
                   = k*stackal
       stackal     = scm(all other al-constants)
       charmax     = scm(charsize,charal)
                     scm = smallest common multiple
       lcaftermarkstack >= 4*ptrsize+max(x-size)
                         = kl*stackelsize          *)
    maxstack    =       1;
    parmal      = stackal;
    parmsize    = stackelsize;
    recal       = stackal;
    filebuffer =        4;
    maxaddr     =  maxint;


type                                                          (*describing:*)
                                                              (*************)


                                                              (*basic symbols*)
                                                              (***************)
      symbol = (ident,intconst,realconst,stringconst,notsy,mulop,addop,relop,
                lparent,rparent,lbrack,rbrack,comma,semicolon,period,arrow,
                colon,becomes,labelsy,constsy,typesy,varsy,funcsy,progsy,
                procsy,setsy,packedsy,arraysy,recordsy,filesy,forwardsy,
                beginsy,ifsy,casesy,repeatsy,whilesy,forsy,withsy,
                gotosy,endsy,elsesy,untilsy,ofsy,dosy,tosy,downtosy,
                thensy,othersy),
      qperator = (mul,rdiv,andop,idiv,imod,plus,minus,orop,ltop,leop,geop,gtop,
                  neop,eqop,inop,noop);
      setofsys = set of symbol;
      chtp = (letter,number,special,illegal,
              chstrquo,chcolon,chperiod,chlt,chgt,chlparen,chspace),

                                                              (*constants*)
                                                              (************)
      cstclass = (reel,pset,strg);
      csp = ^ constant;
      constant = record case cclass: cstclass of
                          reel: (rval: packed array [1..strglgth] of char);
                          pset: (pval: set of setlow..sethigh);
                          strg: (slgth: 0..strglgth;
                                 sval: packed array [1..strglgth] of char)
                        end;

      valu = record case intval: boolean of  (*intval never set nor tested*)
                      true:  (ival: integer);
                      false: (valp: csp)
                    end;

                                                              (*data structures*)
                                                              (******************)
      levrange = 0..maxlevel; addrrange = 0..maxaddr;
      structform = (scalar,subrange,pointer,power,arrays,records,files,
                    tagfld,variant);
      declkind = (standard,declared);
      stp = ^ structure; ctp = ^ identifier;

      structure = packed record
                    marked: boolean;   (*for test phase only*)
                    size: addrrange;
```

```
121                 case form: structform of
122                   scalar:   (case scalkind: declkind of
123                                 declared: (fconst: ctp));
124                   subrange: (rangetype: stp; min,max: valu),
125                   pointer:  (eltype: stp),
126                   power:    (elset: stp);
127                   arrays:   (aeltype,inxtype: stp);
128                   records:  (fstfld: ctp; recvar: stp);
129                   files:    (filtype: stp);
130                   tagfld:   (tagfieldp: ctp; fstvar: stp);
131                   variant:  (nxtvar,subvar: stp; varval: valu)
132                 end;
133
134                                                          (*names*)
135                                                          (*******)
136
137     idclass = (types,konst,vars,field,proc,func);
138     setofids = set of idclass;
139     idkind = (actual,formal);
140     alpha = packed array [1..8] of char;
141
142     identifier = packed record
143                  name: alpha; llink, rlink: ctp;
144                  idtype: stp; next: ctp;
145                  case klass: idclass of
146                    konst: (values: valu);
147                    vars:  (vkind: idkind; vlev: levrange; vaddr: addrrange),
148                    field: (fldaddr: addrrange);
149                    proc,
150                    func:  (case pfdeckind: declkind of
151                              standard: (key: 1..15);
152                              declared: (pflev: levrange; pfname: integer;
153                                         case pfkind: idkind of
154                                          actual: (forwdecl, extern:
155                                                   boolean))) ;
156                  end;
157
158
159     disprange = 0..displimit;
160     where = (blck,crec,vrec,rec);
161
162                                                          (*expressions*)
163                                                          (*************)
164     attrkind = (cst,varbl,expr);
165     vaccess = (drct,indrct,inxd);
166
167     attr = record typtr: stp;
168              case kind: attrkind of
169                cst:   (cval: valu);
170                varbl: (case access: vaccess of
171                          drct: (vlevel: levrange; dplmt: addrrange);
172                          indrct: (idplmt: addrrange))
173              end;
174
175     testp = ^ testpointer;
176     testpointer = packed record
177                     eltl,elt2 : stp;
178                     lasttestp : testp
179                   end;
180
181                                                                (*labels*)
182                                                                (********)
183     lbp = ^ labl;
184     labl = record nextlab: lbp; defined: boolean;
```

```
185                     labval, labname: integer
186             end;
187
188       extfilep = ^filerec;
189       filerec = record filename:alpha; nextfile:extfilep end;
190
191 (*-------------------------------------------------------------------------*)
192
193
194 var
195                                     (*returned by source program scanner
196                                      insymbol:
197                                      **********)
198
199     sy: symbol;                     (*last symbol*)
200     op: operator;                   (*classification of last symbol*)
201     val: valu;                      (*value of last constant*)
202     lgth: integer;                  (*length of last string constant*)
203     id: alpha;                      (*last identifier (possibly truncated)*)
204     kk: 1..8;                       (*nr of chars in last identifier*)
205     ch: char;                       (*last character*)
206     eol: boolean;                   (*end of line flag*)
207
208
209                                     (*counters:*)
210                                     (***********)
211
212     chcnt: integer;                 (*character counter*)
213     lc,ic: addrrange;               (*data location and instruction counter*)
214     linecount. integer;
215          .
216
217                                     (*switches:*)
218                                     (***********)
219
220     dp,                             (*declaration part*)
221     prterr,                         (*to allow forward references in pointer type
222                                       declaration by suppressing error message*)
223     list,prcode,prtables: boolean;  (*output options for
224                                         -- source program listing
225                                         -- printing symbolic code
226                                         -- displaying ident and struct tables
227                                         --> procedure option*)
228     debug: boolean;
229
230
231                                     (*pointers:*)
232                                     (***********)
233     parmptr,
234     intptr,realptr,charptr,
235     boolptr,nilptr,textptr: stp;    (*pointers to entries of standard ids*)
236     utypptr,ucstptr,uvarptr,
237     ufldptr,uprcptr,ufctptr,        (*pointers to entries for undeclared ids*)
238     fwptr: ctp;                     (*head of chain of forw decl type ids*)
239     fextfilep: extfilep;            (*head of chain of external files*)
240     globtestp: testp;               (*last testpointer*)
241
242
243                                     (*bookkeeping of declaration levels:*)
244                                     (************************************)
245
246     level: levrange;                (*current static level*)
247     disx,                           (*level of last id searched by searchid*)
248     top: disprange;                 (*top of display*)
```

```
249
250     display:                          (*where:   means:*)
251       array [disprange] of
252         packed record                 (*=blck:   id is variable id*)
253           fname: ctp; flabel: lbp;    (*=crec:   id is field id in record with*)
254           case occur: where of        (*         constant address*)
255             crec: (clev: levrange;    (*=vrec:   id is field id in record with*)
256                    cdspl: addrrange);(*         variable address*)
257             vrec: (vdspl: addrrange)
258           end;                        (* --> procedure withstatement*)
259
260
261                                       (*error messages:*)
262                                       (*****************)
263
264     errinx: 0..10;                    (*nr of errors in current source line*)
265     errlist:
266       array [1..10] of
267         packed record pos: integer;
268                       nmr: 1..400
269               end;
270
271
272
273                                       (*expression compilation:*)
274                                       (*************************)
275
276     gattr: attr;                      (*describes the expr currently compiled*)
277
278
279                                       (*structured constants:*)
280                                       (************************)
281
282     constbegsys,simptypebegsys,typebegsys,blockbegsys,selectsys,facbegsys,
283     statbegsys,typedels: setofsys;
284     chartp : array[char] of chtp;
285     rw:  array [1..35(*nr. of res. words*)] of alpha;
286     frw: array [1..9] of 1..36(*nr. of res. words + 1*);
287     rsy: array [1..35(*nr. of res. words*)] of symbol;
288     ssy: array [char] of symbol;
289     rop: array [1..35(*nr. of res. words*)] of operator;
290     sop: array [char] of operator;
291     na:  array [1..35] of alpha;
292     mn:  array [0..60] of packed array [1..4] of char;
293     sna: array [1..23] of packed array [1..4] of char;
294     cdx: array [0..60] of -4..+4;
295     pdx: array [1..23] of -7..+7;
296     ordint: array [char] of integer;
297
298     intlabel,mxint10,digmax: integer;
299
300 (*-------------------------------------------------------------------------*)
301
302
303   procedure endofline;
304     var lastpos,freepos,currpos,currnmr,f,k: integer;
305   begin
306     if errinx > 0 then    (*output error messages*)
307       begin write(output,' ****  ':15);
308         lastpos := 0; freepos := 1;
309         for k := 1 to errinx do
310           begin
311             with errlist[k] do
312               begin currpos := pos; currnmr := nmr end;
```

```
313             if currpos = lastpos then write(output,',')
314             else
315               begin
316                 while freepos < currpos do
317                   begin write(output,' '); freepos := freepos + 1 end;
318                 write(output,'^');
319                 lastpos := currpos
320               end;
321             if currnmr < 10 then f := 1
322             else if currnmr < 100 then f := 2
323               else f := 3;
324             write(output,currnmr:f);
325             freepos := freepos + f + 1
326           end;
327         writeln(output); errinx := 0
328       end;
329     linecount := linecount + 1;
330     if list and (not eof(input)) then
331       begin write(output,linecount:6,'  ':2);
332         if dp then write(output,lc:7) else write(output,ic:7);
333         write(output,' ')
334       end;
335     chcnt := 0
336   end  (*endofline*) ;
337
338   procedure error(ferrnr: integer);
339   begin
340     if errinx >= 9 then
341       begin errlist[10].nmr := 255; errinx := 10 end
342     else
343       begin errinx := errinx + 1;
344         errlist[errinx].nmr := ferrnr
345       end;
346     errlist[errinx].pos := chcnt
347   end (*error*) ;
348
349   procedure insymbol;
350     (*read next basic symbol of source program and return its
351     description in the global variables sy, op, id, val and lgth*)
352     label 1,2,3;
353     var i,k: integer;
354         digit: packed array [1..strglgth] of char;
355         string: packed array [1..strglgth] of char;
356         lvp: csp; test: boolean;
357
358     procedure nextch;
359     begin if eol then
360       begin if list then writeln(output); endofline
361       end;
362       if not eof(input) then
363        begin eol := eoln(input); read(input,ch);
364         if list then write(output,ch);
365         chcnt := chcnt + 1
366        end
367       else
368         begin writeln(output,'   *** eof ','encountered');
369           test := false
370         end
371     end;
372
373     procedure options;
374     begin
375       repeat nextch;
376         if ch <> '*' then
```

```
377           begin
378             if ch = 't' then
379               begin nextch; prtables := ch = '+' end
380             else
381               if ch = 'l' then
382                 begin nextch; list := ch = '+';
383                   if not list then writeln(output)
384                 end
385               else
386              if ch = 'd' then
387                begin nextch; debug := ch = '+' end
388               else
389                  if ch = 'c' then
390                    begin nextch; prcode := ch = '+' end;
391             nextch
392           end
393       until ch <> ',';
394     end (*options*) ;
395
396   begin (*insymbol*)
397   1:
398     repeat while (ch = ' ') and not eol do nextch;
399       test := eol;
400       if test then nextch
401     until not test;
402     if chartp[ch] = illegal then
403       begin sy := othersy; op := noop;
404         error(399); nextch
405       end
406     else
407     case chartp[ch] of
408       letter:
409         begin k := 0;
410           repeat
411             if k < 8 then
412              begin k := k + 1; id[k] := ch end ;
413             nextch
414           until chartp[ch] in [special,illegal,chstrquo,chcolon,
415                                chperiod,chlt,chgt,chlparen,chspace];
416           if k >= kk then kk := k
417           else
418             repeat id[kk] := ' '; kk := kk - 1
419             until kk = k;
420           for i := frw[k] to frw[k+1] - 1 do
421             if rw[i] = id then
422               begin sy := rsy[i]; op := rop[i]; goto 2 end;
423             sy := ident; op := noop;
424   2:    end;
425       number:
426         begin op := noop; i := 0;
427           repeat i := i+1; if i<= digmax then digit[i] := ch; nextch
428           until chartp[ch] <> number;
429           if (ch = '.') or (ch = 'e') then
430             begin
431                   k := i;
432                   if ch = '.' then
433                     begin k := k+1; if k <= digmax then digit[k] := ch;
434                       nextch; if ch = '.' then begin ch := ':'; goto 3 end;
435                       if chartp[ch] <> number then error(201)
436                       else
437                         repeat k := k + 1;
438                           if k <= digmax then digit[k] := ch; nextch
439                         until chartp[ch] <>  number
440                     end;
```

```
441                 if ch = 'e' then
442                   begin k := k+1; if k <= digmax then digit[k] := ch;
443                     nextch;
444                     if (ch = '+') or (ch ='-') then
445                       begin k := k+1; if k <= digmax then digit[k] := ch;
446                         nextch
447                       end;
448                     if chartp[ch] <> number then error(201)
449                     else
450                       repeat k := k+1;
451                         if k <= digmax then digit[k] := ch; nextch
452                       until chartp[ch] <> number
453                   end;
454                 new(lvp,reel); sy:= realconst; lvp^.cclass := reel;
455                 with lvp^ do
456                   begin for i := 1 to strglgth do rval[i] := ' ',
457                     if k <= digmax then
458                       for i := 2 to k + 1 do rval[i] := digit[i-1]
459                     else begin error(203); rval[2] := '0';
460                            rval[3] := '.'; rval[4] := '0'
461                          end
462                   end;
463                 val.valp := lvp
464             end
465           else
466   3:        begin
467               if i > digmax then begin error(203); val.ival := 0 end
468               else
469                 with val do
470                   begin ival := 0;
471                     for k := 1 to i do
472                       begin
473                         if ival <= mxint10 then
474                           ival := ival*10+ordint[digit[k]]
475                         else begin error(203); ival := 0 end
476                       end;
477                     sy := intconst
478                   end
479             end
480         end;
481       chstrquo:
482         begin lgth := 0; sy := stringconst;  op := noop;
483           repeat
484             repeat nextch; lgth := lgth + 1;
485                    if lgth <= strglgth then string[lgth] := ch
486             until (eol) or (ch = '''');
487             if eol then error(202) else nextch
488           until ch <> '''';
489           lgth := lgth - 1;   (*now lgth = nr of chars in string*)
490           if lgth = 0 then error(205) else
491           if lgth = 1 then val.ival := ord(string[1])
492           else
493             begin new(lvp,strg); lvp^.cclass:=strg;
494               if lgth > strglgth then
495                 begin error(399); lgth := strglgth end;
496               with lvp^ do
497                 begin slgth := lgth;
498                   for i := 1 to lgth do sval[i] := string[i]
499                 end;
500               val.valp := lvp
501             end
502         end;
503       chcolon:
504         begin op := noop; nextch;
```

```
505           if ch = '=' then
506             begin sy := becomes; nextch end
507           else sy := colon
508         end;
509       chperiod:
510         begin op := noop; nextch;
511           if ch = '.' then
512             begin sy := colon; nextch end
513           else sy := period
514         end;
515       chlt:
516         begin nextch; sy := relop;
517           if ch = '=' then
518             begin op := leop; nextch end
519           else
520             if ch = '>' then
521               begin op := neop; nextch end
522             else op := ltop
523         end;
524       chgt:
525         begin nextch; sy := relop;
526           if ch = '=' then
527             begin op := geop; nextch end
528           else op := gtop
529         end;
530       chlparen:
531        begin nextch;
532          if ch = '*' then
533            begin nextch;
534              if ch = '$' then options;
535              repeat
536                while (ch <> '*') and not eof(input) do nextch;
537                nextch
538              until (ch = ')') or eof(input);
539              nextch; goto 1
540            end;
541          sy := lparent; op := noop
542        end;
543       special:
544         begin sy := ssy[ch]; op := sop[ch];
545           nextch
546         end;
547       chspace: sy := othersy
548     end (*case*)
549   end (*insymbol*) ;
550
551   procedure enterid(fcp: ctp);
552     (*enter id pointed at by fcp into the name-table,
553      which on each declaration level is organised as
554      an unbalanced binary tree*)
555     var nam: alpha; lcp, lcp1: ctp; lleft: boolean;
556   begin nam := fcp^.name;
557     lcp := display[top].fname;
558     if lcp = nil then
559       display[top].fname := fcp
560     else
561       begin
562         repeat lcp1 := lcp;
563           if lcp^.name = nam then   (*name conflict, follow right link*)
564             begin error(101); lcp := lcp^.rlink; lleft := false end
565           else
566             if lcp^.name < nam then
567               begin lcp := lcp^.rlink; lleft := false end
568             else begin lcp := lcp^.llink; lleft := true end
```

```
569           until lcp = nil;
570           if lleft then lcp1^.llink := fcp else lcp1^.rlink := fcp
571         end;
572       fcp^.llink := nil; fcp^.rlink := nil
573     end (*enterid*) ;
574
575     procedure searchsection(fcp: ctp; var fcp1: ctp);
576       (*to find record fields and forward declared procedure id's
577        --> procedure proceduredeclaration
578        --> procedure selector*)
579       label 1;
580     begin
581       while fcp <> nil do
582         if fcp^.name = id then goto 1
583         else if fcp^.name < id then fcp := fcp^.rlink
584           else fcp := fcp^.llink;
585   1:  fcp1 := fcp
586     end (*searchsection*) ;
587
588     procedure searchid(fidcls: setofids; var fcp: ctp);
589       label 1;
590       var lcp: ctp;
591     begin
592       for disx := top downto 0 do
593         begin lcp := display[disx].fname;
594           while lcp <> nil do
595             if lcp^.name = id then
596               if lcp^.klass in fidcls then goto 1
597               else
598                 begin if prterr then error(103);
599                   lcp := lcp^.rlink
600                 end
601             else
602               if lcp^.name < id then
603                 lcp := lcp^.rlink
604               else lcp := lcp^.llink
605         end;
606       (*search not successful; suppress error message in case
607        of forward referenced type id in pointer type definition
608        --> procedure simpletype*)
609       if prterr then
610         begin error(104);
611           (*to avoid returning nil, reference an entry
612            for an undeclared id of appropriate class
613            --> procedure enterundecl*)
614           if types in fidcls then lcp := utypptr
615           else
616             if vars in fidcls then lcp := uvarptr
617             else
618               if field in fidcls then lcp := ufldptr
619               else
620                 if konst in fidcls then lcp := ucstptr
621                 else
622                   if proc in fidcls then lcp := uprcptr
623                   else lcp := ufctptr;
624         end;
625   1:  fcp := lcp
626     end (*searchid*) ;
627
628     procedure getbounds(fsp: stp; var fmin,fmax: integer);
629       (*get internal bounds of subrange or scalar type*)
630       (*assume fsp<>intptr and fsp<>realptr*)
631     begin
632       fmin := 0; fmax := 0;
```

```
633     if fsp <> nil then
634     with fsp^ do
635       if form = subrange then
636         begin fmin := min.ival; fmax := max.ival end
637       else
638           if fsp = charptr then
639             begin fmin := ordminchar; fmax := ordmaxchar
640             end
641           else
642             if fconst <> nil then
643               fmax := fconst^.values.ival
644   end (*getbounds*) ;
645
646   function alignquot(fsp: stp): integer;
647   begin
648     alignquot := 1;
649     if fsp <> nil then
650       with fsp^ do
651         case form of
652           scalar:    if fsp=intptr then alignquot := intal
653                      else if fsp=boolptr then alignquot := boolal
654                      else if scalkind=declared then alignquot := intal
655                      else if fsp=charptr then alignquot := charal
656                      else if fsp=realptr then alignquot := realal
657                      else (*parmptr*) alignquot := parmal;
658           subrange: alignquot := alignquot(rangetype);
659           pointer:  alignquot := adral;
660           power:    alignquot := setal;
661           files:    alignquot := fileal;
662           arrays:   alignquot := alignquot(aeltype);
663           records:  alignquot := recal;
664           variant,tagfld: error(501)
665         end
666   end (*alignquot*);
667
668   procedure align(fsp: stp; var flc: integer);
669     var k,l: integer;
670   begin
671     k := alignquot(fsp);
672     l := flc-1;
673     flc := l + k  -  (k+l) mod k
674   end (*align*);
675
676   procedure printtables(fb: boolean);
677     (*print data structure and name table*)
678     var i, lim: disprange;
679
680     procedure marker;
681       (*mark data structure entries to avoid multiple printout*)
682       var i: integer;
683
684       procedure markctp(fp: ctp); forward;
685
686       procedure markstp(fp: stp);
687         (*mark data structures, prevent cycles*)
688       begin
689         if fp <> nil then
690           with fp^ do
691             begin marked := true;
692               case form of
693               scalar:   ;
694               subrange: markstp(rangetype);
695               pointer:  (*don't mark eltype: cycle possible; will be marked
696                         anyway, if fp = true*) ;
```

```
697             power:    markstp(elset) ;
698             arrays:   begin markstp(aeltype); markstp(inxtype) end;
699             records:  begin markctp(fstfld); markstp(recvar) end;
700             files:    markstp(filtype);
701             tagfld:   markstp(fstvar),
702             variant:  begin markstp(nxtvar); markstp(subvar) end
703             end (*case*)
704           end (*with*)
705       end (*markstp*);
706
707       procedure markctp;
708       begin
709         if fp <> nil then
710           with fp^ do
711             begin markctp(llink); markctp(rlink);
712               markstp(idtype)
713             end
714       end (*markctp*);
715
716     begin (*marker*)
717       for i := top downto lim do
718         markctp(display[i].fname)
719     end (*marker*);
720
721     procedure followctp(fp: ctp); forward;
722
723     procedure followstp(fp: stp);
724     begin
725       if fp <> nil then
726         with fp^ do
727           if marked then
728             begin marked := false; write(output,' ':4,ord(fp):6,size:10);
729               case form of
730               scalar:   begin write(output,'scalar':10);
731                           if scalkind = standard then
732                             write(output,'standard':10)
733                           else write(output,'declared':10,' ':4,ord(fconst):6);
734                           writeln(output)
735                         end;
736               subrange: begin
737                           write(output,'subrange':10,' ':4,ord(rangetype):6);
738                           if rangetype <> realptr then
739                             write(output,min.ival,max.ival)
740                           else
741                             if (min.valp <> nil) and (max.valp <> nil) then
742                               write(output,' ',min.valp^.rval:9,
743                                     ' ',max.valp^.rval:9);
744                           writeln(output); followstp(rangetype);
745                         end;
746               pointer:  writeln(output,'pointer':10,' ':4,ord(eltype):6);
747               power:    begin writeln(output,'set':10,' ':4,ord(elset):6);
748                           followstp(elset)
749                         end;
750               arrays:   begin
751                           writeln(output,'array':10,' ':4,ord(aeltype):6,' ':4,
752                             ord(inxtype):6);
753                           followstp(aeltype); followstp(inxtype)
754                         end;
755               records:  begin
756                           writeln(output,'record':10,' ':4,ord(fstfld):6,' ':4,
757                             ord(recvar):6); followctp(fstfld);
758                           followstp(recvar)
759                         end;
760               files:    begin write(output,'file':10,' ':4,ord(filtype):6);
```

```
761                       followstp(filtype)
762                     end;
763           tagfld:   begin writeln(output,'tagfld':10,' ':4,ord(tagfieldp):6,
764                       ' ':4,ord(fstvar):6);
765                     followstp(fstvar)
766                     end;
767           variant:  begin writeln(output,'variant':10,' ':4,ord(nxtvar):6,
768                       ' ':4,ord(subvar):6,varval.ival);
769                     followstp(nxtvar); followstp(subvar)
770                     end
771         end (*case*)
772       end (*if marked*)
773   end (*followstp*);
774
775   procedure followctp;
776     var i: integer;
777   begin
778     if fp <> nil then
779       with fp^ do
780         begin write(output,' ':4,ord(fp):6,' ',name:9,' ':4,ord(llink):6,
781           ' ':4,ord(rlink):6,' ':4,ord(idtype):6);
782           case klass of
783             types: write(output,'type':10);
784             konst: begin write(output,'constant':10,' ':4,ord(next):6);
785                      if idtype <> nil then
786                        if idtype = realptr then
787                          begin
788                            if values.valp <> nil then
789                              write(output,' ',values.valp^.rval:9)
790                          end
791                        else
792                          if idtype^.form = arrays then  (*stringconst*)
793                            begin
794                              if values.valp <> nil then
795                                begin write(output,' ');
796                                  with values.valp^ do
797                                    for i := 1 to slgth do
798                                      write(output,sval[i])
799                                end
800                            end
801                          else write(output,values.ival)
802                    end;
803             vars:  begin write(output,'variable':10);
804                      if vkind = actual then write(output,'actual':10)
805                      else write(output,'formal':10);
806                      write(output,' ':4,ord(next):6,vlev,' ':4,vaddr:6 );
807                    end;
808             field: write(output,'field':10,' ':4,ord(next):6,' ':4,fldaddr:6);
809             proc,
810             func:  begin
811                      if klass = proc then write(output,'procedure':10)
812                      else write(output,'function':10);
813                      if pfdeckind = standard then
814                        write(output,'standard':10, key:10)
815                      else
816                        begin write(output,'declared':10,' ':4,ord(next):6);
817                          write(output,pflev,' ':4,pfname:6);
818                          if pfkind = actual then
819                            begin write(output,'actual':10);
820                              if forwdecl then write(output,'forward':10)
821                              else write(output,'notforward':10);
822                              if extern then write(output,'extern':10)
823                              else write(output,'not extern':10);
824                            end
```