Thus the reference in a Pascal program to a file variable for example input↑, will be a direct reference to the contents of one of these locations.

P-CODE INSTRUCTION SET

The following table describes the complete instruction set showing the parameters and also the effect of the execution of each instruction on the stack. Only a brief description of each instruction is given here – a detailed version is given in Chapter 11.

Instruction	Operation on stack		Parameters if present	Description of instruction
	Before	After		
abi	(i)	i		Absolute value of integer
abr	(r)	r		Absolute value of real
adi	(i,i)	i		Adds two integers on the top of the stack and leaves an integer result
adr	(r,r)	r		Adds two reals on the top of the stack and leaves a real result
chk C	No change			Checks value is between upper and lower bounds
chr	(i)	c		Converts integer to character
csp	special		Q	Call standard procedure
cup	special		PQ	Call user procedure
dec C	(x)	x	Q	Decrement
dif	(s,s)	s		Set difference
dvi	(i,i)	i		Integer division
dvr	(r,r)	r		Real division
ent	special		PQ	Enter block
eof	(a)	b		Test on end of file
equ C	(x,x)	b	Q	Compare on equal
fjp	(b)			False jump
flo	(i,r)	r,r		Float next to the top
flt	(i)	r		Float top of the stack
geq C	(x,x)	b	Q	Compare on greater or equal
inc C	(x)	x	Q	Increment
ind C	(a)	x	Q	Indexed fetch
inn	(i,s)	b		Test set membership
int	(s,s)	s		Set intersection
ior	(b,b)	b		Boolean inclusive OR
ixa	(a,i)	a	Q	Compute indexed address
lao		a	Q	Load base-level address
lca		a	Q	Load address of constant
lci		x	PQ	Load constant indirect – assembler generated
lda		a	PQ	Load address with level P
ldc C		x	Q	Load constant
ldo C		x	Q	Load contents of base-level address
leq C	(x,x)	b	Q	Compare on less than or equal
les C	(x,x)	b	Q	Compare on less than
lod C		x	PQ	Load contents of address
mod	(i,i)	i		Modulo
mov	(a,a)		Q	Move
mpi	(i,i)	i		Integer multiplication
mpr	(r,r)	r		Real multiplication

Instruction	Operation on stack		Parameters if present	Description of instruction
	Before	After		
mst	special		P	Mark stack
neq C	(x,x)	b	Q	Compare on not equal
ngi	(i)	i		Integer sign inversion
ngr	(r)	r		Real sign inversion
not	(b)	b		Boolean not
odd	(i)	b		Test on odd
ord C	(x)	i		Convert to integer
ret C	special			Return from block
sbi	(i,i)	i		Integer subtraction
sbr	(r,r)	r		Real subtraction
sgs	(i)	s		Generate singleton set
sqi	(i)	i		Square integer
sqr	(r)	r		Square real
sro C	(x)		Q	Store at base level address
sto C	(a,x)			Store at base-level address
stp	No effect			Stop
str C	(x)		PQ	Store at level P
trc	(i)	i		Truncation
ujc	No effect			Error in case statement
ujp	No effect		Q	Unconditional jump
uni	(s,s)	s		Set union
xjp	(i)		Q	Indexed jump

Key to effect on stack:

a	address
b	boolean
c	character
i	integer
r	real
s	set
x	any of the above types

The C parameter denotes one of the primitive types.

Assembler/Interpreter Listing

```
  1 (*Assembler and interpretor of Pascal code*)
  2 (*K. Jensen, N. Wirth, Ch. Jacobi, ETH May 76*)
  3
  4 program pcode(input,output,prd,prr);
  5
  6 (* Note for the implementation.
  7    ============================
  8 This interpretor is written for the case where all the fundamental types
  9 take one storage unit.
 10 In an actual implementation, the handling of the sp pointer has to take
 11 into account the fact that the types may have lengths different from one;
 12 in push and pop operations the sp has to be increased and decreased not
 13 by 1, but by a number depending on the type concerned.
 14 However, where the number of units of storage has been computed by the
 15 compiler, the value must not be corrected, since the lengths of the types
 16 involved have already been taken into account.
 17
 18                                                                    *)
 19
 20
 21
 22 label 1;
 23 const codemax     = 8650;
 24       pcmax       = 17500;
 25       maxstk      = 13650; (* size of variable store *)
 26       overi       = 13655; (* size of integer constant table = 5 *)
 27       overr       = 13660; (* size of real constant table = 5 *)
 28       overs       = 13730; (* size of set constant table = 70 *)
 29       overb       = 13820;
 30       overm       = 18000;
 31       maxstr,     = 18001;
 32       largeint    = 26144;
 33       begincode   = 3;
 34       inputadr    = 5;
 35       outputadr   = 6;
 36       prdadr      = 7;
 37       prradr      = 8;
 38       duminst     = 62;
 39
 40 type  bit4       =-0..15;
 41       bit6       = 0..127;
 42       bit20      = -26143..26143;
 43       datatype   = (undef,int,real,bool,setf,adr,mark,car);
 44       address    = -1..maxstr;
 45       beta       = packed array[1..25] of char; (*error message*)
 46
 47 var   code       : array[0..codemax] of   (* the program *)
 48                      packed record  opl    :bit6;
 49                                     pl     :bit4;
 50                                     ql     :bit20;
 51                                     op2    :bit6;
 52                                     p2     :bit4;
 53                                     q2     :bit20
 54                                end;
 55       pc         : 0..pcmax;          (*program address register*)
 56       op : bit6; p : bit4; q : bit20;
```



```
953                     end;
954
955          46 (*int*): begin  sp := sp-1;
956                      store[sp].va := store[sp].va * store[sp+1].va
957                     end;
958
959          47 (*uni*): begin  sp := sp-1;
960                      store[sp].va := store[sp].va + store[sp+1].va
961                     end;
962
963          48 (*inn*): begin
964                      sp := sp - 1; i := store[sp].vi;
965                      store[sp].vb := i in store[sp+1].vs;
966                     end;
967
968          49 (*mod*): begin  sp := sp-1;
969                      store[sp].vi := store[sp].vi mod store[sp+1].vi
970                     end;
971
972          50 (*odd*): store[sp].vb := odd(store[sp].vi);
973
974          51 (*mpi*): begin  sp := sp-1;
975                      store[sp].vi := store[sp].vi * store[sp+1].vi
976                     end;
977
978          52 (*mpr*): begin  sp := sp-1;
979                      store[sp].vr := store[sp].vr * store[sp+1].vr
980                     end;
981
982          53 (*dvi*): begin  sp := sp-1;
983                      store[sp].vi := store[sp].vi div store[sp+1].vi
984                     end;
985
986          54 (*dvr*): begin  sp := sp-1;
987                      store[sp].vr := store[sp].vr / store[sp+1].vr
988                     end;
989
990          55 (*mov*): begin i1 := store[sp-1].va;
991                      i2 := store[sp].va; sp := sp-2;
992                      for i := 0 to q-1 do store[i1+i] := store[i2+i]
993                      (* q is a number of storage units *)
994                     end;
995
996          56 (*lca*): begin  sp := sp+1;
997                      store[sp].va := q;
998                     end;
999
1000         100,101,102,103,104,
1001         57 (*dec*): store[sp].vi := store[sp].vi-q;
1002
1003         58 (*stp*): interpreting := false;
1004
1005         59 (*ord*): (*only used to change the tagfield*)
1006                     begin
1007                     end;
1008
1009         60 (*chr*): begin
1010                     end;
1011
1012         61 (*ujc*): errori(' case - error          ');
1013   end
1014 end; (*while interpreting*)
1015
1016 1 :
```

```
57
58      store       : array [0..overm] of
59                    record case datatype of
60                                 int      :(vi :integer),
61                                 reel     :(vr :real),
62                                 bool     :(vb :boolean),
63                                 sett     :(vs :set of 0..47);
64                                 car      :(vc :char);
65                                 adr      :(va :address);
66                                   (*address in store*)
67                                 mark     :(vm :integer)
68                           end;
69      mp,sp,np,ep : address;  (* address registers *)
70      (*mp  points to beginning of a data segment
71        sp  points to top of the stack
72        ep  points to the maximum extent of the stack
73        np  points to top of the dynamically allocated area*)
74
75      interpreting: boolean;
76      prd,prr     : text;(*prd for read only, prr for write only *)
77
78      instr       : array[bit6] of alfa; (* mnemonic instruction codes *)
79      cop         : array[bit6] of integer;
80      sptable     : array[0..20] of alfa; (*standard functions and procedures*)
81
82      (*locally used for interpreting one instruction*)
83      ad,ad1      : address;
84      b           : boolean;
85      i,j,i1,i2   : integer;
86      c           : char;
87
88 (*-------------------------------------------------------------------*)
89
90 procedure load;
91    const maxlabel = 1050;
92    type  labelst  = (entered,defined); (*label situation*)
93          labelrg  = 0..maxlabel;       (*label range*)
94          labelrec = rucord
95                       val: address;
96                        st: labelst
97                     end;
98    var  icp,rcp,scp,bcp,mcp  : address;  (*pointers to next free position*)
99         word : array[1..10] of char; i   : integer;  ch  : char;
100        labeltab: array[labelrg] of labelrec;
101        labelvalue: address;
102
103   procedure init;
104      var i: integer;
105   begin instr[ 0]:='lod       ';      instr[ 1]:='ldo       ';
106         instr[ 2]:='str       ';      instr[ 3]:='sro       ';
107         instr[ 4]:='lda       ';      instr[ 5]:='lao       ';
108         instr[ 6]:='sto       ';      instr[ 7]:='ldc       ';
109         instr[ 8]:='...       ';      instr[ 9]:='ind       ';
110         instr[10]:='inc       ';      instr[11]:='mst       ';
111         instr[12]:='cup       ';      instr[13]:='ent       ';
112         instr[14]:='ret       ';      instr[15]:='csp       ';
113         instr[16]:='ixa       ';      instr[17]:='equ       ';
114         instr[18]:='neq       ';      instr[19]:='geq       ';
115         instr[20]:='grt       ';      instr[21]:='leq       ';
116         instr[22]:='lae       ';      instr[23]:='ujp       ';
117         instr[24]:='fjp       ';      instr[25]:='xjp       ';
118         instr[26]:='chk       ';      instr[27]:='eof       ';
119         instr[28]:='adi       ';      instr[29]:='adr       ';
120         instr[30]:='sbi       ';      instr[31]:='sbr       ';
```



```
121         instr[32]:='sgs       ';      instr[33]:='flt       ';
122         instr[34]:='flo       ';      instr[35]:='trc       ';
123         instr[36]:='ngi       ';      instr[37]:='ngr       ';
124         instr[38]:='sqi       ';      instr[39]:='sqr       ';
125         instr[40]:='abi       ';      instr[41]:='abr       ';
126         instr[42]:='not       ';      instr[43]:='and       ';
127         instr[44]:='ior       ';      instr[45]:='dif       ';
128         instr[46]:='int       ';      instr[47]:='uni       ';
129         instr[48]:='inn       ';      instr[49]:='mod       ';
130         instr[50]:='odd       ';      instr[51]:='mpi       ';
131         instr[52]:='mpr       ';      instr[53]:='dvi       ';
132         instr[54]:='dvr       ';      instr[55]:='mov       ';
133         instr[56]:='lca       ';      instr[57]:='dec       ';
134         instr[58]:='stp       ';      instr[59]:='ord       ';
135         instr[60]:='chr       ';      instr[61]:='ujc       ';
136
137         sptable[ 0]:='get       ';    sptable[ 1]:='put       ';
138         sptable[ 2]:='rst       ';    sptable[ 3]:='rln       ';
139         sptable[ 4]:='new       ';    sptable[ 5]:='wln       ';
140         sptable[ 6]:='wrs       ';    sptable[ 7]:='eln       ';
141         sptable[ 8]:='wri       ';    sptable[ 9]:='wrr       ';
142         sptable[10]:='wrc       ';    sptable[11]:='rdi       ';
143         sptable[12]:='rdr       ';    sptable[13]:='rdc       ';
144         sptable[14]:='sin       ';    sptable[15]:='cos       ';
145         sptable[16]:='exp       ';    sptable[17]:='log       ';
146         sptable[18]:='sqt       ';    sptable[19]:='atn       ';
147         sptable[20]:='eov       ';
148
149         cop[ 0] := 105;  cop[ 1] :=  65;
150         cop[ 2] :=  70;  cop[ 3] :=  75;
151         cop[ 6] :=  80;  cop[ 9] :=  85;
152         cop[10] :=  90;  cop[26] :=  95;
153         cop[57] := 100;
154
155         pc  := begincode;
156         icp := maxstk + 1;
157         rcp := overi + 1;
158         scp := overr + 1;
159         bcp := overs + 2;
160         mcp := overb + 1;
161         for i:= 1 to 10 do word[i]:= ' ';
162         for i:= 0 to maxlabel do
163            with labeltab[i] do begin val:=-1; st:= entered end;
164         reset(prd);
165   end,(*init*)
166
167   procedure error1(string: beta); (*error in loading*)
168   begin writeln;
169      write(string);
170      halt
171   end; (*error1*)
172
173   procedure update(x: labelrg); (*when a label definition lx is found*)
174      var curr,succ: -1..pcmax;  (*resp. current element and successor elemunt
175                                   of a list of future references*)
176          endlist: boolean;
177   begin
178      if labeltab[x].st=defined then error1(' duplicated label        ')
179      else begin
180             if labeltab[x].val<>-1 then (*forward reference(s)*)
181             begin curr:= labeltab[x].val; endlist:= false;
182                while not endlist do
183                      with code[curr div 2] do
184                      begin
```

```
185                     if odd(curr) then begin succ:= q2;
186                                           q2:= labelvalue
187                                     end
188                                  else begin succ:= q1;
189                                           q1:= labelvalue
190                                     end;
191                     if succ=-1 then endlist:= true
192                                else curr:= succ
193                  end;
194            end;
195            labeltab[x].st := defined;
196            labeltab[x].val:= labelvalue;
197         end
198   end,(*update*)
199
200   procedure assemble; forward;
201
202   procedure generate;(*generate segment of code*)
203      var x: integer; (* label number *)
204          again: boolean;
205   begin
206      again := true;
207      while again do
208           begin read(prd,ch),(* first character of line*)
209                case ch of
210                     'i': readln(prd),
211                     'l': begin read(prd,x);
212                                if not eoln(prd) then read(prd,ch);
213                                if ch='=' then read(prd,labelvalue)
214                                          else labelvalue:= pc;
215                                update(x); readln(prd);
216                          end;
217                     'q': begin again := false; readln(prd) end;
218                     ' ': begin read(prd,ch); assemble end
219                end;
220           end
221   end; (*generate*)
222
223   procedure assemble; (*translate symbolic code into machine code and store*)
224      label 1;          (*goto 1 for instructions without code generation*)
225      var name :alfa;  b :boolean;  r :real;  s :set of 0..58;
226          ch :char;  i,sl,lb,ub :integer;
227
228      procedure lookup(x: labelrg); (* search in label table*)
229      begin case labeltab[x].st of
230                entered: begin q := labeltab[x].val;
231                           labeltab[x].val := pc
232                         end;
233                defined: q:= labeltab[x].val
234           end(*case label..*)
235      end;(*lookup*)
236
237      procedure labelsearch;
238         var x: labelrg;
239      begin while (ch<>'l') and not eoln(prd) do read(prd,ch);
240           read(prd,x); lookup(x)
241      end;(*labelsearch*)
242
243      procedure getname;
244      begin  word[1] := ch;
245         read(prd,word[2],word[3]);
246         if not eoln(prd) then read(prd,ch) (*next character*);
247         pack(word,1,name)
248      end; (*getname*)
```



```
249
250      procedure typesymbol;
251        var i: integer;
252      begin
253        if ch <> 'i' then
254          begin
255            case ch of
256              'a': i := 0;
257              'r': i := 1;
258              's': i := 2;
259              'b': i := 3;
260              'c': i := 4;
261            end;
262            op := cop[op]+i;
263          end;
264      end (*typesymbol*) ;
265
266   begin  p := 0;  q := 0;  op := 0;
267      getname;
268      instr[duminst] := name;
269      while instr[op]<>name do op := op+1;
270      if op = duminst then error1(' illegal instruction     ');
271
272      case op of  (* get parameters p,q *)
273
274          (*equ,neq,geq,grt,leq,les*)
275          17,18,19,
276          20,21,22: begin case ch of
277                              'a': ; (*p = 0*)
278                              'i': p := 1;
279                              'r': p := 2;
280                              'b': p := 3;
281                              's': p := 4;
282                              'c': p := 6;
283                              'm': begin p := 5;
284                                         read(prd,q)
285                                   end
286                    end
287                end;
288
289          (*lod,str*)
290          0,2: begin typesymbol; read(prd,p,q)
291               end;
292
293          4  (*lda*): read(prd,p,q);
294
295          12 (*cup*): begin read(prd,p); labelsearch end;
296
297          11 (*mst*): read(prd,p);
298
299          14 (*ret*): case ch of
300                           'p': p:=0;
301                           'i': p:=1;
302                           'r': p:=2;
303                           'c': p:=3;
304                           'b': p:=4;
305                           'e': p:=5
306                      end;
307
308          (*lao,ixa,mov*)
309          5,16,55: read(prd,q),
310
311          (*ldo,sro,ind,inc,dec*)
312          1,3,9,10,57: begin typesymbol; read(prd,q)
```

```
313                 end;
314
315     (*ujp,fjp,xjp*)
316     23,24,25: labelsearch;
317
318     13 (*onc*): begin read(prd,p), labelsearch end;
319
320     15 (*cap*): begin for i:=1 to 9 do read(prd,ch); getname;
321                   while name<>sptable[q] do  q := q+1
322                 end;
323
324     7 (*ldc*): begin case ch of  (*get q*)
325                   'i': begin  p := 1;  read(prd,i);
326                          if abs(i)>=largeint then
327                          begin  op := 8;
328                            store[icp].vi := i;  q := maxstk;
329                            repeat  q := q+1  until store[q].vi=i;
330                            if q=icp then
331                            begin  icp := icp+1;
332                              if icp=overi then
333                                errorl(' integer table overflow  ');
334                            end
335                          end  else q := i
336                        end;
337
338                   'r': begin  op := 8; p := 2;
339                          read(prd,r);
340                          store[rcp].vr := r;  q := overi;
341                          repeat  q := q+1  until store[q].vr=r;
342                          if q=rcp then
343                          begin  rcp := rcp+1;
344                            if rcp = overr then
345                              errorl(' real table overflow     ');
346                          end
347                        end;
348
349                   'n': ; (*p,q = 0*)
350
351                   'b': begin p := 3;  read(prd,q)  end;
352
353                   'c': begin p := 6;
354                          repeat read(prd,ch); until ch <> ' ';
355                          if ch <> '''' then
356                            errorl(' illegal character       ');
357                          read(prd,ch);  q := ord(ch);
358                          read(prd,ch);
359                          if ch <> '''' then
360                            errorl(' illegal character       ');
361                        end;
362                   '(': begin  op := 8;  p := 4;
363                          s := [ ];  read(prd,ch),
364                          while ch<>')' do
365                          begin read(prd,si,ch); s := s + [si]
366                          end;
367                          store[scp].vs := s;  q := overr;
368                          repeat  q := q+1  until store[q].vs=s;
369                          if q=scp then
370                          begin  scp := scp+1;
371                            if scp=overs then
372                              errorl(' set table overflow      ');
373                          end
374                        end
375                 end (*case*)
376           end;
```



```
377
378           26 (*chk*): begin typesymbol;
379                       read(prd,lb,ub);
380                       if op = 95 then q := lb
381                       else
382                       begin
383                         store[bcp-1].vi := lb; store[bcp].vi := ub;
384                         q := overs;
385                         repeat  q := q+2
386                         until (store[q-1].vi=lb)and (store[q].vi=ub),
387                         if q=bcp then
388                         begin  bcp := bcp+2;
389                           if bcp=overb then
390                             errorl(' boundary table overflow ');
391                         end
392                       end;
393
394           56 (*lca*): begin
395                       if mcp + 16 >= overm then
396                         errorl(' multiple table overflow '),
397                       mcp := mcp+16;
398                       q := mcp;
399                       for i := 0 to 15 (*stringlgth*) do
400                       begin read(prd,ch);
401                         store[q+i].vc := ch
402                       end;
403                     end;
404
405           6 (*sto*): typesymbol;
406
407           27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,
408           48,49,50,51,52,53,54,58:  ;
409
410           (*ord,chr*)
411           59,60: goto 1;
412
413           61 (*ujc*): ; (*must have same length as ujp*)
414
415       end; (*case*)
416
417       (* store instruction *)
418       with code[pc div 2] do
419         if odd(pc) then
420         begin  op2 := op; p2 := p; q2 := q
421         end  else
422         begin  op1 := op; p1 := p; q1 := q
423         end;
424       pc := pc+1;
425       1: readln(prd);
426     end; (*assemble*)
427
428 begin (*load*)
429     init;
430     generate;
431     pc := 0;
432     generate;
433 end; (*load*)
434
435 (*--------------------------------------------------------------------------*)
436
437 procedure pmd;
438     var s :integer; i: integer;
439
440
```

```
                    8: errori(' readln on prr file      ')
                end;
                sp:= sp-1
            end;
 4 (*new*): begin ad:= np-store[sp].va;
            (*top of stack gives the length in units of storage *)
                if ad <= ep then
                  errori(' store overflow           ');
                np:= ad; ad:= store[sp-1].va;
                store[ad].va := np;
                sp:=sp-2
            end;
 5 (*wln*): begin case store[sp].va of
                    5: errori(' writeln on input file   '),
                    6: writeln(output);
                    7: errori(' writeln on prd file     ');
                    8: writeln(prr)
                end;
                sp:= sp-1
            end;
 6 (*wrs*): case store[sp].va of
                5: errori(' write on input file     ');
                6: writestr(output);
                7: errori(' write on prd file       ');
                8: writestr(prr)
            end;
 7 (*eln*): begin case store[sp].va of
                    5: line:= eoln(input);
                    6: errori(' eoln output file        '),
                    7: line:=eoln(prd);
                    8: errori(' eoln on prr file        ')
                end;
                store[sp].vb := line
            end;
 8 (*wri*): begin case store[sp].va of
                    5: errori(' write on input file     ');
                    6: write(output,
                         store[sp-2].vi: store[sp-1].vi);
                    7: errori(' write on prd file       ');
                    8: write(prr,
                         store[sp-2].vi: store[sp-1].vi)
                end;
                sp:=sp-3
            end;
 9 (*wrr*): begin case store[sp].va of
                    5: errori(' write on input file     '),
                    6: write(output,
                         store[sp-2].vr: store[sp-1].vi);
                    7: errori(' write on prd file       ');
                    8: write(prr,
                         store[sp-2].vr: store[sp-1].vi)
                end;
                sp:=sp-3
            end;
10(*wrc*): begin case store[sp].va of
                    5: errori(' write on input file     ');
                    6: write(output,store[sp-2].vc:
                         store[sp-1].vi);
                    7: errori(' write on prd file       ');
                    8: write(prr,chr(store[sp-2].vi):
                         store[sp-1].vi);
                end;
                sp:=sp-3
            end;
```



```
11(*rdi*): case store[sp].va of
                5: readi(input),
                6: errori(' read on output file     ');
                7: readi(prd),
                8: errori(' read on prr file        ')
            end;
12(*rdr*): case store[sp].va of
                5: readr(input);
                6: errori(' read on output file     '),
                7: readr(prd),
                8: errori(' read on prr file        ')
            end;
13(*rdc*): case store[sp].va of
                5: readc(input);
                6: errori(' read on output file     '),
                7: readc(prd);
                8: errori(' read on prr file        ')
            end;
14(*sin*): store[sp].vr:= sin(store[sp].vr);
15(*cos*): store[sp].vr:= cos(store[sp].vr);
16(*exp*): store[sp].vr:= exp(store[sp].vr);
17(*log*): store[sp].vr:= ln(store[sp].vr);
18(*sqt*): store[sp].vr:= sqrt(store[sp].vr);
19(*atn*): store[sp].vr:= arctan(store[sp].vr);
20(*sav*): begin ad:=store[sp].va;
             store[ad].va := np;
             sp:= sp-1
           end;
    end;(*case q*)
end;(*callsp*)

begin (* main *)
  rewrite(prr);
  load; (* assembles and stores code *)
  writeln(output); (* for testing *)
  pc := 0; sp := -1; mp := 0; np := maxstk+1; ep := 5;
  store[inputadr].vc := input^;
  store[prdadr].vc := prd^;
  interpreting := true;

  while interpreting do
  begin
    (*fetch*)
    with code[pc div 2] do
      if odd(pc) then
      begin op := op2; p := p2, q := q2
      end else
      begin op := op1; p := p1; q := q1
      end;
    pc := pc+1;

    (*execute*)
    case op of

        105,106,107,108,109,
        0 (*lod*): begin  ad := base(p) + q;
                     sp := sp+1;
                     store[sp] := store[ad]
                   end;

        65,66,67,68,69,
        1 (*ldo*): begin
                     sp := sp+1;
                     store[sp] := store[q]
```

```
441     procedure pt;
442     begin  write(s:6);
443        if abs(store[s].vi) < maxint then write(store[s].vi)
444        else write('too big ');
445        s := s - 1;
446        i := i + 1;
447        if i = 4 then
448           begin writeln(output); i := 0 end;
449     end; (*pt*)
450
451  begin
452     write(' pc =',pc-1:5,' op =',op:3,'  sp =',sp:5,'  mp =',mp:5,
453           '  np =',np:5);
454     writeln; writeln('-----------------------------------');
455
456     s := sp; i := 0;
457     while s>=0 do pt;
458     s := maxstk;
459     while s>=np do pt;
460  end; (*pmd*)
461
462  procedure errori(string; beta);
463  begin writeln; writeln(string);
464        pmd; goto 1
465  end;(*errori*)
466
467  function base(ld :integer):address;
468     var ad :address;
469  begin  ad := mp;
470     while ld>0 do
471     begin  ad := store[ad+1].va;  ld := ld-1
472     end;
473     base := ad
474  end; (*base*)
475
476  procedure compare;
477  (*comparing is only correct if result by comparing integers will be*)
478  begin
479    l1 := store[sp].va;
480    l2 := store[sp+1].va;
481    i := 0; b := true;
482    while b and (i<>q) do
483      if store[l1+i].vi = store[l2+i].vi then i := i+1
484      else b := false
485  end; (*compare*)
486
487  procedure callsp;
488     var line: boolean; adptr,adelnt: address;
489         i: integer;
490
491     procedure readi(var f:text);
492        var ad: address;
493     begin ad:= store[sp-1].va;
494           read(f,store[ad].vi);
495           store[store[sp].va].vc := f^;
496           sp:= sp-2
497     end;(*readi*)
498
499     procedure readr(var f: text);
500        var ad: address;
501     begin ad:= store[sp-1].va;
502           read(f,store[ad].vr);
503           store[store[sp].va].vc := f^;
504           sp:= sp-2
```



```
505     end;(*readr*)
506
507     procedure readc(var f: text);
508        var c: char; ad: address;
509     begin read(f,c);
510           ad:= store[sp-1].va;
511           store[ad].vc := c;
512           store[store[sp].va].vc := f^;
513           store[store[sp].va].vi := ord(f^);
514           sp:= sp-2
515     end;(*readc*)
516
517     procedure writestr(var f: text);
518        var i,j,k: integer;
519            ad: address.
520     begin ad:= store[sp-3].va;
521           k := store[sp-2].vi; j := store[sp-1].vi;
522           (* j and k are numbers of characters *)
523           if k>j then for i:=1 to k-j do write(f,' ')
524                  else j:= k;
525           for i := 0 to j-1 do write(f,store[ad+i].vc);
526           sp:= sp-4
527     end;(*writestr*)
528
529     procedure getfile(var f: text);
530        var ad: address;
531     begin ad:=store[sp].va;
532           get(f), store[ad].vc := f^;
533           sp:=sp-1
534     end;(*getfile*)
535
536     procedure putfile(var f: text);
537        var ad: address;
538     begin ad:= store[sp].va;
539           f^:= store[ad].vc; put(f);
540           sp:= sp-1;
541     end;(*putfile*)
542
543  begin (*callsp*)
544        case q of
545           0 (*get*): case store[sp].va of
546                         5: getfile(input);
547                         6: errori(' get on output file      ');
548                         7: getfile(prd);
549                         8: errori(' get on prr file         ')
550                      end;
551           1 (*put*): case store[sp].va of
552                         5: errori(' put on read file        ');
553                         6: putfile(output);
554                         7: errori(' put on prd file         ');
555                         8: putfile(prr)
556                      end;
557           2 (*rst*): begin
558                        (*for testphase*)
559                        np := store[sp].va; sp := sp-1
560                      end;
561           3 (*rln*): begin case store[sp].va of
562                                5: begin readln(input);
563                                     store[inputadr].vc := input^
564                                   end;
565                                6: errori(' readln on output file   ');
566                                7: begin readln(input);
567                                     store[inputadr].vc := input^
568                                   end;
```

```
                end;

      70,71,72,73,74,
      2 (*str*): begin  store[base(p)+q] := store[sp];
                 sp := sp-1
                end;

      75,76,77,78,79,
      3 (*sro*): begin  store[q] := store[sp];
                 sp := sp-1
                end;

      4 (*lda*): begin sp := sp+1;
                 store[sp].va := base(p) + q
                end;

      5 (*lao*): begin sp := sp+1;
                 store[sp].va := q
                end;

      80,81,82,83,84,
      6 (*sto*): begin
                 store[store[sp-1].va] := store[sp];
                 sp := sp-2;
                end;

      7 (*ldc*): begin sp := sp+1;
                 if p=1 then
                 begin store[sp].vi := q;
                 end else
                     if p = 6 then store[sp].vc := chr(q)
                     else
                       if p = 3 then store[sp].vb := q = 1
                       else (* load nil *) store[sp].va := maxstr
                end;

      8 (*lci*): begin sp := sp+1;
                 store[sp] := store[q]
                end;

      85,86,87,88,89,
      9 (*ind*): begin ad := store[sp].va + q;
                 (* q is a number of storage units *)
                 store[sp] := store[ad]
                end;

      90,91,92,93,94,
      10 (*inc*): store[sp].vi := store[sp].vi+q;

      11 (*mst*): begin (*p=level of calling procedure minus level of called
                          procedure + 1;  set dl and sl, increment sp*)
                  (* then length of this element is
                     max(intsize,realsize,boolsize,charsize,ptrsize *)
                  store[sp+2].vm := base(p);
                  (* the length of this element is ptrsize *)
                  store[sp+3].vm := mp;
                  (* idem *)
                  store[sp+4].vm := sp;
                  (* idem *)
                  sp := sp+5
                end;

      12 (*cup*): begin (*p=no of locations for parameters, q=entry point*)
                  mp := sp-(p+4);
                  store[mp+4].vm := pc;
                  pc := q
                end;

      13 (*ent*): if p = 1 then
                   begin sp := mp + q; (*q = length of dataseg*)
                     if sp > np then errori(' store overflow          ');
                   end
                  else
                   begin sp := sp+q;
                     if sp > np then errori(' store overflow          ');
                   end;
                   (*q = max space required on stack*)

      14 (*ret*): begin case p of
                            0: sp:= mp-1;
                            1,2,3,4,5: sp:= mp
                       end;
                       pc := store[mp+4].vm;
                       sp := store[mp+3].vm;
                       mp:= store[mp+2].vm;
                  end;

      15 (*csp*): callsp;

      16 (*ixa*): begin
                  i := store[sp].vi;
                  sp := sp-1;
                  store[sp].va := q*i+store[sp].va;
                 end;

      17 (*equ*): begin  sp := sp-1;
                  case p of
                    1: store[sp].vb := store[sp].vi = store[sp+1].vi;
                    0: store[sp].vb := store[sp].va = store[sp+1].va;
                    6: store[sp].vb := store[sp].vc = store[sp+1].vc;
                    2: store[sp].vb := store[sp].vr = store[sp+1].vr;
                    3: store[sp].vb := store[sp].vb = store[sp+1].vb;
                    4: store[sp].vb := store[sp].vs = store[sp+1].vs;
                    5: begin  compare;
                         store[sp].vb := b;
                       end;
                  end; (*case p*)
                 end;

      18 (*neq*): begin  sp := sp-1;
                  case p of
                    0: store[sp].vb := store[sp].va <> store[sp+1].va;
                    1: store[sp].vb := store[sp].vi <> store[sp+1].vi;
                    6: store[sp].vb := store[sp].vc <> store[sp+1].vc;
                    2: store[sp].vb := store[sp].vr <> store[sp+1].vr;
                    3: store[sp].vb := store[sp].vb <> store[sp+1].vb;
                    4: store[sp].vb := store[sp].vs <> store[sp+1].vs;
                    5: begin  compare;
                         store[sp].vb := not b;
                       end
                  end; (*case p*)
                 end;

      19 (*geq*): begin  sp := sp-1;
                  case p of
                    0: errori(' <,<=,>,>= for address   ');
                    1: store[sp].vb := store[sp].vi >= store[sp+1].vi;
                    6: store[sp].vb := store[sp].vc >= store[sp+1].vc;
```

```
825                   2: store[sp].vb := store[sp].vr >= store[sp+1].vr;
826                   3: store[sp].vb := store[sp].vb >= store[sp+1].vb;
827                   4: store[sp].vb := store[sp].vs >= store[sp+1]vs;
828                   5: begin compare;
829                        store[sp].vb := b or
830                          (store[i1+i].vi >= store[i2+i].vi)
831                      end
832                 end; (*case p*)
833               end;
834
835       20 (*grt*): begin  sp := sp-1;
836                   case p of
837                     0: errori(' <,<=,>,>= for address   ');
838                     1: store[sp].vb := store[sp].vi > store[sp+1].vi;
839                     6: store[sp].vb := store[sp].vc > store[sp+1].vc;
840                     2: store[sp].vb := store[sp].vr > store[sp+1].vr;
841                     3: store[sp].vb := store[sp].vb > store[sp+1].vb;
842                     4: errori(' set inclusion          ');
843                     5: begin  compare;
844                          store[sp].vb := not b and
845                            (store[i1+i].vi > store[i2+i].vi)
846                        end
847                 end; (*case p*)
848               end;
849
850       21 (*leq*): begin  sp := sp-1;
851                   case p of
852                     0: errori(' <,<=,>,>= for address   ');
853                     1: store[sp].vb := store[sp].vi <= store[sp+1].vi;
854                     6: store[sp].vb := store[sp].vc <= store[sp+1].vc;
855                     2: store[sp].vb := store[sp].vr <= store[sp+1].vr;
856                     3: store[sp].vb := store[sp].vb <= store[sp+1].vb;
857                     4: store[sp].vb := store[sp].vs <= store[sp+1].vs,
858                     5: begin  compare;
859                          store[sp].vb := b or
860                            (store[i1+i].vi <= store[i2+i].vi)
861                        end;
862                 end; (*case p*)
863               end;
864
865       22 (*les*): begin  sp := sp-1;
866                   case p of
867                     0: errori(' <,<=,>,>= for address   ');
868                     1: store[sp].vb := store[sp].vi < store[sp+1].vi;
869                     6: store[sp].vb := store[sp].vc < store[sp+1].vc;
870                     2: store[sp].vb := store[sp].vr < store[sp+1].vr;
871                     3: store[sp].vb := store[sp].vb < store[sp+1].vb;
872                     5: begin  compare;
873                          store[sp].vb := not b and
874                            (store[i1+i].vi < store[i2+i].vi)
875                        end
876                 end; (*case p*)
877               end;
878
879       23 (*ujp*): pc := q;
880
881       24 (*fjp*): begin  if not store[sp].vb then pc := q;
882                   sp := sp-1
883                 end;
884
885       25 (*xjp*): begin
886                   pc := store[sp].vi + q;
887                   sp := sp-1
888                 end;
```



```
889
890        95 (*chka*): if (store[sp].va < np) or
891                       (store[sp].va > (maxstr-q)) then
892                     errori(' bad pointer value       ');
893
894       96,97,98,99,
895       26 (*chk*): if (store[sp].vi < store[q-1].vi) or
896                     (store[sp].vi > store[q].vi) then
897                   errori(' value out of range      '),
898
899       27 (*eof*): begin  i := store[sp].vi;
900                   if i=inputadr then
901                   begin store[sp].vb := eof(input),
902                   end else errori(' code in error          ')
903                 end;
904
905       28 (*adi*): begin  sp := sp-1;
906                   store[sp].vi := store[sp].vi + store[sp+1].vi
907                 end;
908
909       29 (*adr*): begin  sp := sp-1;
910                   store[sp].vr := store[sp].vr + store[sp+1].vr
911                 end;
912
913       30 (*sbi*): begin sp := sp-1;
914                   store[sp].vi := store[sp].vi - store[sp+1].vi
915                 end;
916
917       31 (*sbr*): begin  sp := sp-1;
918                   store[sp].vr := store[sp].vr - store[sp+1].vr
919                 end;
920
921       32 (*sgs*): store[sp].vs := [store[sp].vi];
922
923       33 (*flt*): store[sp].vr := store[sp].vi;
924
925       34 (*flo*): store[sp-1].vr := store[sp-1].vi;
926
927       35 (*trc*): store[sp].vi := trunc(store[sp].vr);
928
929       36 (*ngi*): store[sp].vi := -store[sp].vi;
930
931       37 (*ngr*): store[sp].vr := -store[sp].vr;
932
933       38 (*sqi*): store[sp].vi := sqr(store[sp].vi);
934
935       39 (*sqr*): store[sp].vr := sqr(store[sp].vr);
936
937       40 (*abi*): store[sp].vi := abs(store[sp].vi);
938
939       41 (*abr*): store[sp].vr := abs(store[sp].vr);
940
941       42 (*not*): store[sp].vb := not store[sp].vb;
942
943       43 (*and*): begin  sp := sp-1;
944                   store[sp].vb := store[sp].vb and store[sp+1].vb
945                 end;
946
947       44 (*ior*): begin  sp := sp-1;
948                   store[sp].vb := store[sp].vb or store[sp+1].vb
949                 end;
950
951       45 (*dif*): begin  sp := sp-1;
952                   store[sp].vs := store[sp].vs - store[sp+1].vs
```

TYPES

Types are represented internally by the type *structure* [118-32].

All types use the *size* field, which contains the run-time store-size needed to hold an object of that type. (The *marked* field is used only by the procedure *printtables* [676-845] when printing out the compiler tables, if the *t* option is switched on.)

All other fields depend on the *form* of the type, if it is a pointer, or an array, and so on.

Scalar

A scalar type is either *declared*, that is, an enumeration, in which case *fconst* points to the last identifier in the list (they are linked together by their *next* field) [1061]; or it is *standard*, when the type is *integer*, *real*, or *char* (*boolean* is *declared*). These latter four can be distinguished by comparing the pointer value to the *structure* with one of the four pointers *intptr*, *realptr*, *charptr*, *boolptr*, which are initialised [3646 *et seq.*] (see for example [652-7]).

Subrange

Here *rangetype* points to the type of which this is a subrange (for example, *integer* in *1 . . 10*); and *min* and *max* hold the minimum and maximum values (*1* and *10* in the above case).

Pointer

Eltype points to the type pointed at (*integer* in ↑*integer*).

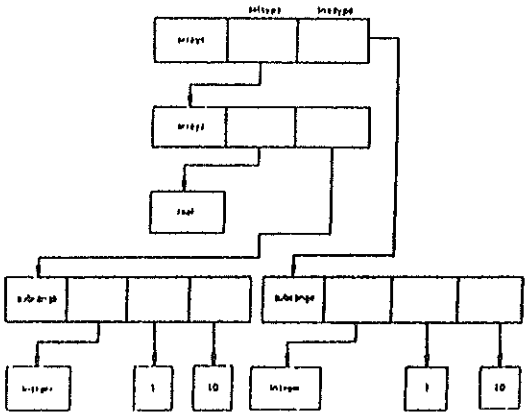
Power

For sets, *elset* points to the element type of the set (for example, *char* in set of *char*).

Arrays

Inxtype points to the index type of the array and *aeltype* to the element type, (e.g. *char* and *integer* respectively in array*[char]* of *integer*).

A multi-dimensional array, like array *[1. .10, 1. .10]* of *real* is treated identically to array *[1. .10]* of array *[1. .10]* of *real* so here would give



Files

Filtype points to the file type (for example *integer* in file of *integer*).

Records

Fstfld points to the first field of the record, the other fields being linked to the binary tree described before.

Recvar points to a *structure* of form *tagfld* representing the variant part of the record (it is nil if there is no variant part).

A *tagfld* has two fields: *tagfldp* points to the *identifier* for the tag field, and *fstvar* points to a list of *structures* of form *variant*, each representing one of the case labels.

As an example:

```
type r= record i: integer;
          case b: colour of
            red, blue: (j: integer);
            green: (k: integer;
                    case c: boolean of
                      true: (a: real)
          end;
```

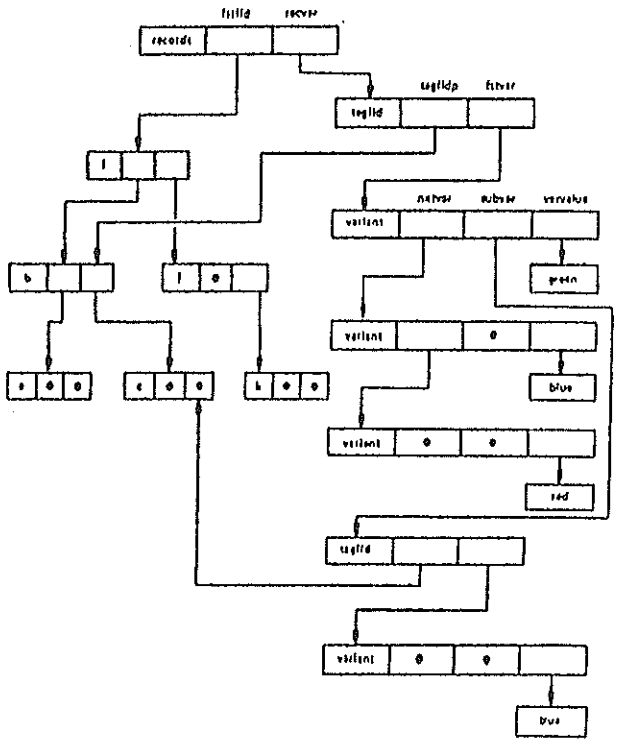


TABLE OF CONTENTS

Compiler Listing 3
Assembler/Interpreter Listing 67

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)

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    setlow      =      0;
51    sethigh     =     47;
52    ordmaxchar  =     63;
53    ordminchar  =      0;
54    maxint      =  32767;
55    leftermarkstack = 5;
```

```
57    (* stackelsize = minimum size for 1 stackelement
58                    = k*stackal
59       stackal     = scm(all other sl-constants)
60       charmax     = scm(charsize,charal)
61                     scm = smallest common multiple
62       lcaftermarkstack >= 4*ptrsize+max(x-size)
63                          = kl*stackelsize          *)
64    maxstack    =       1;
65    parmal      = stackal;
66    parmsize    = stackelsize;
67    recal       = stackal;
68    filebuffer  =       4;
69    maxaddr     =  maxint;
70
71
72
73  type                                                    (*describing:*)
74                                                          (***************)
75
76
77                                                          (*basic symbols*)
78                                                          (***************)
79
80      symbol = (ident,intconst,realconst,stringconst,notsy,mulop,addop,relop,
81                lparent,rparent,lbrack,rbrack,comma,semicolon,period,arrow,
82                colon,becomes,labelsy,constsy,typesy,varsy,funcsy,progsy,
83                procsy,setsy,packedsy,arraysy,recordsy,filesy,forwardsy,
84                beginsy,ifsy,casesy,repeatsy,whilesy,forsy,withsy,
85                gotosy,endsy,elsesy,untilsy,ofsy,dosy,tosy,downtosy,
86                thensy,othersy);
87      operator = (mul,rdiv,andop,idiv,imod,plus,minus,orop,ltop,leop,geop,gtop,
88                  neop,eqop,inop,noop);
89      setofsys = set of symbol;
90      chtp = (letter,number,special,illegal,
91              chstrquo,chcolon,chperiod,chlt,chgt,chlparen,chspace),
92
93                                                          (*constants*)
94                                                          (***********)
95
96      cstclass = (real,pset,strg);
97      csp = ^ constant;
98      constant = record case cclass: cstclass of
99                          real: (rval: packed array [1..strglgth] of char);
100                         pset: (pval: set of setlow..sethigh);
101                         strg: (slgth: 0..strglgth;
102                                sval: packed array [1..strglgth] of char)
103                       end;
104
105     valu = record case intval: boolean of  (*intval never set nor tested*)
106                    true:  (ival: integer);
107                    false: (valp: csp)
108                  end;
109
110                                                         (*data structures*)
111                                                         (*****************)
112     levrange = 0..maxlevel; addrrange = 0..maxaddr;
113     structform = (scalar,subrange,pointer,power,arrays,records,files,
114                   tagfld,variant);
115     declkind = (standard,declared);
116     stp = ^ structure; ctp = ^ identifier;      !
117
118     structure = packed record
119                   marked: boolean;   (*for test phase only*)
120                   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                     elt1,elt2 : stp;
178                     lasttestp : testp
179                   end;
180
181                                                         (*labels*)
182                                                         (********)
183     lbp = ^ labl;
184     labl = record nxtlab: 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:    name:*)
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,ic: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; dobug := ch = '+' end
388                else
389                  if ch = 'c' then
390         if ch<>'*' begin nextch; prcode := ch = '+' end;
391         while 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 <= maxint10 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: sy := ssy[ch]; op := sop[ch];
544         nextch
545       end;
546       chspace: sy := othersy
547     end (*case*)
548   end (*insymbol*) ;
549
550   procedure entorid(fcp: ctp);
551     (*enter id pointed at by fcp into the name-table,
552      which on each declaration level is organised as
553      an unbalanced binary tree*)
554     var nam: alpha; lcp, lcp1: ctp; lleft: boolean;
555   begin nam := fcp^.name;
556     lcp := display[top].fname;
557     if lcp = nil then
558       display[top].fname := fcp
559     else
560       begin
561         repeat lcp1 := lcp;
562           if lcp^.name = nam then   (*name conflict, follow right link*)
563             begin error(101); lcp := lcp^.rlink; lleft := false end
564           else
565             if lcp^.name < nam then
566               begin lcp := lcp^.rlink; lleft := false end
567             else begin lcp := lcp^.llink; lleft := true end
568
```

```
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 := filesal;
662           arrays:   alignquot := alignquot(eltype);
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+1) 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 mark(fp: ctp); forward;
681
682     procedure markctp(fp: ctp);
683       (*mark data structure entries to avoid multiple printout*)
684       var i: integer;
685
686       procedure markstp(fp: stp); forward;
687
688       procedure markstp(fp: stp);
689         (*mark data structures, prevent cycles*)
690       begin
691         if fp <> nil then
692           with fp^ do
693             begin marked := true;
694               case form of
695               scalar:   ;
696               subrange: markstp(rangetype);
697               pointer:  (*don't mark eltype: cycle possible; will be marked
698                          anyway, if fp = true*) ;
```

```
825                         else write(output,'formal':10)
826                       end
827                   end
828           end (*case*);
829           writeln(output);
830           followctp(llink); followctp(rlink);
831           followctp(idtype)
832         end (*with*)
833     end (*followctp*);
834
835   begin (*printtables*)
836     writeln(output); writeln(output); writeln(output);
837     if fb then lim := 0
838     else begin lim := top; write(output,' local') end;
839     writeln(output,' tables '); writeln(output);
840     marker;
841     for i := top downto lim do
842       followctp(display[i].fname);
843     writeln(output);
844     if not eol then write(output,' ':chcnt+16)
845   end (*printtables*);
846
847   procedure genlabel(var nxtlab: integer);
848   begin intlabel := intlabel + 1;
849     nxtlab := intlabel
850   end (*genlabel*);
851
852   procedure block(fsys: setofsys; fsy: symbol; fprocp: ctp);
853     var lsy: symbol; test: boolean;
854
855     procedure skip(fsys: setofsys);
856       (*skip input string until relevant symbol found*)
857     begin
858       if not eof(input) then
859         begin while not(sy in fsys) and (not eof(input)) do insymbol;
860           if not (sy in fsys) then insymbol
861         end
862     end (*skip*) ;
863
864     procedure constant(fsys: setofsys; var fsp: stp; var fvalu: valu);
865       var lsp: stp; lcp: ctp; sign: (none,pos,neg);
866           lvp: csp; i: 2..strglgth;
867     begin lsp := nil; fvalu.ival := 0;
868       if not(sy in constbegsys) then
869         begin error(50); skip(fsys+constbegsys) end;
870       if sy in constbegsys then
871         begin
872           if sy = stringconstsy then
873             begin
874               if lgth = 1 then lsp := charptr
875               else
876                 begin
877                   new(lsp,arrays);
878                   with lsp^ do
879                     begin aeltype := charptr; inxtype := nil;
880                       size := lgth*charsize; form := arrays
881                     end
882                 end;
883               fvalu := val; insymbol
884             end
885           else
886             begin
887               sign := none;
888               if (sy = addop) and (op in [plus,minus]) then
889                 begin if op = plus then sign := pos else sign := neg;
890                   insymbol
891                 end;
892               if sy = ident then
893                 begin searchid([konst],lcp);
894                   with lcp^ do
895                     begin lsp := idtype; fvalu := values end;
896                   if sign <> none then
897                     if lsp = intptr then
898                       begin if sign = neg then fvalu.ival := -fvalu.ival end
899                     else
900                       if lsp = realptr then
901                         begin
902                           if sign = neg then
903                             begin new(lvp,real);
904                               if fvalu.valp^.rval[1] = '-' then
905                                 lvp^.rval[1] := '+'
906                               else lvp^.rval[1] := '-';
907                               for i := 2 to strglgth do
908                                 lvp^.rval[i] := fvalu.valp^.rval[i];
909                               fvalu.valp := lvp;
910                             end
911                         end
912                       else error(105);
913                   insymbol;
914                 end
915               else
916                 if sy = intconst then
917                   begin if sign = neg then val.ival := -val.ival;
918                     lsp := intptr; fvalu := val; insymbol
919                   end
920                 else
921                   if sy = realconst then
922                     begin if sign = neg then val.valp^.rval[1] := '-';
923                       lsp := realptr; fvalu := val; insymbol
924                     end
925                   else
926                     begin error(106); skip(fsys) end
927           end;
928         if not (sy in fsys) then
929           begin error(6); skip(fsys) end
930         end;
931     fsp := lsp
932   end (*constant*) ;
933
934   function equalbounds(fsp1,fsp2: stp): boolean;
935     var lmin1,lmin2,lmax1,lmax2: integer;
936   begin
937     if (fsp1=nil) or (fsp2=nil) then equalbounds := true
938     else
939       begin
940         getbounds(fsp1,lmin1,lmax1);
941         getbounds(fsp2,lmin2,lmax2);
942         equalbounds := (lmin1=lmin2) and (lmax1=lmax2)
943       end
944   end (*equalbounds*) ;
945
946   function comptypes(fsp1,fsp2: stp) : boolean;
947     (*decide whether structures pointed at by fsp1 and fsp2 are compatible*)
948     var nxt1,nxt2: ctp; comp: boolean;
949       ltestp1,ltestp2 : testp;
950   begin
951     if fsp1 = fsp2 then comptypes := true
952     else
```

```
697             power:    marketp(elset) ;
698             arrays:   begin marketp(aeltype); marketp(inxtype) end;
699             records:  begin marketp(fstfld); marketp(recvar) end;
700             files:    marketp(filtype);
701             tagfld:   marketp(fstvar);
702             variant:  begin marketp(nxtvar); marketp(subvar) end
703             end (*case*)
704           end (*with*)
705     end (*marketp*);
706
707     procedure marketp;
708     begin
709       if fp <> nil then
710         with fp^ do
711           begin marketp(llink); marketp(rlink);
712             marketp(idtype)
713           end
714     end (*marketp*);
715
716   begin (*marker*)
717     for i := top downto lim do
718       marketp(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 sigth 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
```

```
953       if (fsp1 <> nil) and (fsp2 <> nil) then
954         if fsp1^.form = fsp2^.form then
955           case fsp1^.form of
956             scalar:
957               comptypes := false;
958               (* identical scalars declared on different levels are
959                not recognized to be compatible*)
960             subrange:
961               comptypes := comptypes(fsp1^.rangetype,fsp2^.rangetype);
962             pointer:
963                 begin
964                   comp := false; ltestp1 := globtestp;
965                   ltestp2 := globtestp;
966                   while ltestp1 <> nil do
967                     with ltestp1^ do
968                       begin
969                         if (elt1 = fsp1^.eltype) and
970                            (elt2 = fsp2^.eltype) then comp := true;
971                         ltestp1 := lasttestp
972                       end;
973                   if not comp then
974                     begin new(ltestp1);
975                       with ltestp1^ do
976                         begin elt1 := fsp1^.eltype;
977                           elt2 := fsp2^.eltype;
978                           lasttestp := globtestp
979                         end;
980                       globtestp := ltestp1;
981                       comp := comptypes(fsp1^.eltype,fsp2^.eltype)
982                     end;
983                   comptypes := comp; globtestp := ltestp2
984                 end;
985             power:
986               comptypes := comptypes(fsp1^.elset,fsp2^.elset);
987             arrays:
988               begin
989                 comp := comptypes(fsp1^.aeltype,fsp2^.aeltype)
990                     and comptypes(fsp1^.inxtype,fsp2^.inxtype);
991                 comptypes := comp and (fsp1^.size = fsp2^.size) and
992                     equalbounds(fsp1^.inxtype,fsp2^.inxtype)
993               end;
994             records:
995               begin nxt1 := fsp1^.fstfld; nxt2 := fsp2^.fstfld; comp:=true;
996                 while (nxt1 <> nil) and (nxt2 <> nil) do
997                   begin comp:=comp and comptypes(nxt1^.idtype,nxt2^.idtype),
998                     nxt1 := nxt1^.next; nxt2 := nxt2^.next
999                   end;
1000                comptypes := comp and (nxt1 = nil) and (nxt2 = nil)
1001                          and(fsp1^.recvar = nil)and(fsp2^.recvar = nil)
1002              end;
1003              (*identical records are recognized to be compatible
1004               iff no variants occur*)
1005            files:
1006              comptypes := comptypes(fsp1^.filtype,fsp2^.filtype)
1007          end (*case*)
1008        else (*fsp1^.form <> fsp2^.form*)
1009          if fsp1^.form = subrange then
1010            comptypes := comptypes(fsp1^.rangetype,fsp2)
1011          else
1012            if fsp2^.form = subrange then
1013              comptypes := comptypes(fsp1,fsp2^.rangetype)
1014            else comptypes := false
1015      else comptypes := true
1016  end (*comptypes*) ;
```



```
1017
1018  function string(fsp: stp) : boolean;
1019  begin string := false;
1020    if fsp <> nil then
1021      if fsp^.form = arrays then
1022        if comptypes(fsp^.aeltype,charptr) then string := true
1023  end (*string*) ;
1024
1025  procedure typ(fsys: setofsys; var fsp: stp; var fsize: addrrange);
1026    var lsp,lsp1,lsp2: stp; oldtop: disprange; lcp: ctp;
1027        lsize,disp1: addrrange; lmin,lmax: integer;
1028
1029    procedure simpletype(fsys:setofsys; var fsp:stp; var fsize:addrrange);
1030      var lsp,lsp1: stp; lcp,lcp1: ctp; ttop: disprange;
1031          lcnt: integer; lvalu: valu;
1032    begin fsize := 1;
1033      if not (sy in simptypebegsys) then
1034        begin error(1); skip(fsys + simptypebegsys) end;
1035      if sy in simptypebegsys then
1036        begin
1037          if sy = lparent then
1038            begin ttop := top;   (*decl. consts local to innermost block*)
1039              while display[top].occur <> blck do top := top - 1;
1040              new(lsp,scalar,declared);
1041              with lsp^ do
1042                begin size := intsize; form := scalar;
1043                  scalkind := declared
1044                end;
1045              lcp1 := nil; lcnt := 0;
1046              repeat insymbol;
1047                if sy = ident then
1048                  begin new(lcp,konst);
1049                    with lcp^ do
1050                      begin name := id; idtype := lsp; next := lcp1;
1051                        values.ival := lcnt; klass := konst
1052                      end;
1053                    enterid(lcp);
1054                    lcnt := lcnt + 1;
1055                    lcp1 := lcp; insymbol
1056                  end
1057                else error(2);
1058                if not (sy in fsys + [comma,rparent]) then
1059                  begin error(6); skip(fsys + [comma,rparent]) end
1060              until sy <> comma;
1061              lsp^.fconst := lcp1; top := ttop;
1062              if sy = rparent then insymbol else error(4)
1063            end
1064          else
1065            begin
1066              if sy = ident then
1067                begin searchid([types,konst],lcp);
1068                  insymbol;
1069                  if lcp^.klass = konst then
1070                    begin new(lsp,subrange);
1071                      with lsp^, lcp^ do
1072                        begin rangetype := idtype; form := subrange;
1073                          if string(rangetype) then
1074                            begin error(148); rangetype := nil end;
1075                          min := values; size := intsize
1076                        end;
1077                      if sy = colon then insymbol else error(5);
1078                      constant(fsys,lsp1,lvalu);
1079                      lsp^.max := lvalu;
1080                      if lsp^.rangetype <> lsp1 then error(107)
```

```
1721              begin name := id; idtype := nil;
1722                extern := false; pflev := level; genlabel(lbname);
1723                pfdeckind := declared; pfkind := actual; pfname := lbname;
1724                if fsy = procsy then klass := proc
1725                else klass := func
1726              end;
1727            enterid(lcp)
1728          end
1729        else
1730          begin lcp1 := lcp^.next;
1731            while lcp1 <> nil do
1732              begin
1733                with lcp1^ do
1734                  if klass = vars then
1735                    if idtype <> nil then
1736                      begin lcm := vaddr + idtype^.size;
1737                        if lcm > lc then lc := lcm
1738                      end;
1739                lcp1 := lcp1^.next
1740              end
1741          end;
1742        insymbol
1743      end
1744    else
1745      begin error(2); lcp := ufctptr end;
1746    oldlev := level; oldtop := top;
1747    if level < maxlevel then level := level + 1 else error(251);
1748    if top < displimit then
1749      begin top := top + 1;
1750        with display[top] do
1751          begin
1752            if forw then fname := lcp^.next
1753            else fname := nil;
1754            flabel := nil;
1755            occur := blck
1756          end
1757      end
1758    else error(250);
1759    if fsy = procsy then
1760      begin parameterlist([semicolon],lcp1);
1761        if not forw then lcp^.next := lcp1
1762      end
1763    else
1764      begin parameterlist([semicolon,colon],lcp1);
1765        if not forw then lcp^.next := lcp1;
1766        if sy = colon then
1767          begin insymbol;
1768            if sy = ident then
1769              begin if forw then error(122);
1770                searchid([types],lcp1);
1771                lsp := lcp1^.idtype;
1772                lcp^.idtype := lsp;
1773                if lsp <> nil then
1774                  if not (lsp^.form in [scalar,subrange,pointer]) then
1775                    begin error(120); lcp^.idtype := nil end;
1776                insymbol
1777              end
1778            else begin error(2); skip(fsys + [semicolon]) end
1779          end
1780        else
1781          if not forw then error(123)
1782      end;
1783    if sy = semicolon then insymbol else error(14);
1784    if sy = forwardsy then
```



```
1785      begin
1786        if forw then error(161)
1787        else lcp^.forwdecl := true;
1788        insymbol;
1789        if sy = semicolon then insymbol else error(14);
1790        if not (sy in fsys) then
1791          begin error(6); skip(fsys) end
1792      end
1793    else
1794      begin lcp^.forwdecl := false; mark(markp);
1795        repeat block(fsys,semicolon,lcp);
1796          if sy = semicolon then
1797            begin if prtables then printtables(false); insymbol;
1798              if not (sy in [beginsy,procsy,funcsy]) then
1799                begin error(6); skip(fsys) end
1800            end
1801          else error(14)
1802        until (sy in [beginsy,procsy,funcsy]) or eof(input);
1803        release(markp); (* return local entries on runtime heap *)
1804      end;
1805    level := oldlev; top := oldtop; lc := llc;
1806  end (*procdeclaration*) ;
1807
1808  procedure body(fsys: setofsys);
1809    const cstoccmax=65; clxmax=1000;
1810    type oprange = 0..63;
1811    var
1812        llcp:ctp; saveid:alpha;
1813        cstptr: array [1..cstoccmax] of csp,
1814        cstptrix: 0..cstoccmax;
1815        (*allows referencing of noninteger constants by an index
1816         (instead of a pointer), which can be stored in the p2-field
1817         of the instruction record until writeout.
1818         --> procedure load, procedure writeout*)
1819        i, entname, segsize: integer;
1820        stacktop, topnew, topmax: integer;
1821        lcmax,llc1: addrrange; lcp: ctp;
1822        llp: lbp;
1823
1824
1825    procedure mes(i: integer);
1826    begin topnew := topnew + cdx[i]*maxstack;
1827      if topnew > topmax then topmax := topnew
1828    end;
1829
1830    procedure putic;
1831    begin if ic mod 10 = 0 then writeln(prr,'i',ic:5) end;
1832
1833    procedure gen0(fop: oprange);
1834    begin
1835      if prcode then begin putic; writeln(prr,mn[fop]:4) end;
1836      ic := ic + 1; mes(fop)
1837    end (*gen0*) ;
1838
1839    procedure gen1(fop: oprange; fp2: integer);
1840      var k: integer;
1841    begin
1842      if prcode then
1843        begin putic; write(prr,mn[fop]:4);
1844          if fop = 30 then
1845            begin writeln(prr,sna[fp2]:12);
1846              topnew := topnew + pdx[fp2]*maxstack;
1847              if topnew > topmax then topmax := topnew
1848            end
```

```
1849          else
1850            begin
1851              if fop = 38 then
1852                 begin write(prr,'''');
1853                   with cstptr[fp2]^ do
1854                   begin
1855                     for k := 1 to slgth do write(prr,sval[k]:1);
1856                     for k := slgth+1 to strglgth do write(prr,' ');
1857                   end;
1858                   writeln(prr,'''')
1859                 end
1860              else if fop = 42 then writeln(prr,chr(fp2))
1861                   else writeln(prr,fp2:12);
1862              mes(fop)
1863            end
1864        end;
1865      ic := ic + 1
1866    end (*gen1*) ;
1867
1868    procedure gen2(fop: oprange; fp1,fp2: integer);
1869      var k : integer;
1870    begin
1871      if prcode then
1872        begin putic; write(prr,mn[fop]:4);
1873          case fop of
1874            45,50,54,56:
1875              writeln(prr,' ',fp1:3,fp2:8);
1876            47,48,49,52,53,55:
1877              begin write(prr,chr(fp1));
1878                if chr(fp1) = 'm' then write(prr,fp2:11);
1879                writeln(prr)
1880              end;
1881            51:
1882              case fp1 of
1883                1: writeln(prr,'i ',fp2);
1884                2: begin write(prr,'r ');
1885                     with cstptr[fp2]^ do
1886                       for k := 1 to strglgth do write(prr,rval[k]);
1887                     writeln(prr)
1888                   end;
1889                3: writeln(prr,'b ',fp2);
1890                4: writeln(prr,'n');
1891                6: writeln(prr,'c ''':3,chr(fp2),'''');
1892                5: begin write(prr,'(');
1893                     with cstptr[fp2]^ do
1894                       for k := setlow to sethigh do
1895                         if k in pval then write(prr,k:3);
1896                     writeln(prr,')')
1897                   end
1898              end
1899          end;
1900        end;
1901      ic := ic + 1; mes(fop)
1902    end (*gen2*) ;
1903
1904    procedure gentypindicator(fsp: stp);
1905    begin
1906      if fsp<>nil then
1907        with fsp^ do
1908          case form of
1909           scalar: if fsp=intptr then write(prr,'i')
1910                   else
1911                     if fsp=boolptr then write(prr,'b')
1912                     else
```



```
1913                     if fsp=charptr then write(prr,'c')
1914                     else
1915                       if scalkind = declared then write(prr,'i')
1916                       else write(prr,'r');
1917            subrange: gentypindicator(rangetype);
1918            pointer: write(prr,'a');
1919            power:   write(prr,'s');
1920            records,arrays: write(prr,'m');
1921            files,tagfld,variant: error(500)
1922          end
1923    end (*typindicator*);
1924
1925    procedure gen0t(fop: oprange; fsp: stp);
1926    begin
1927      if prcode then
1928        begin putic;
1929          write(prr,mn[fop]:4);
1930          gentypindicator(fsp);
1931          writeln(prr);
1932        end;
1933      ic := ic + 1; mes(fop)
1934    end (*gen0t*);
1935
1936    procedure gen1t(fop: oprange; fp2: integer; fsp: stp);
1937    begin
1938      if prcode then
1939        begin putic;
1940          write(prr,mn[fop]:4);
1941          gentypindicator(fsp);
1942          writeln(prr,fp2:11)
1943        end;
1944      ic := ic + 1; mes(fop)
1945    end (*gen1t*);
1946
1947    procedure gen2t(fop: oprange; fp1,fp2: integer; fsp: stp);
1948    begin
1949      if prcode then
1950        begin putic;
1951          write(prr,mn[fop]: 4);
1952          gentypindicator(fsp);
1953          writeln(prr,fp1:3+5*ord(abs(fp1)>99),fp2:8);
1954        end;
1955      ic := ic + 1; mes(fop)
1956    end (*gen2t*);
1957
1958    procedure load;
1959    begin
1960      with gattr do
1961        if typtr <> nil then
1962          begin
1963            case kind of
1964              cst:   if (typtr^.form = scalar) and (typtr <> realptr) then
1965                       if typtr = boolptr then gen2(51(*ldc*),3,cval.ival)
1966                       else
1967                         if typtr=charptr then
1968                           gen2(51(*ldc*),6,cval.ival)
1969                         else gen2(51(*ldc*),1,cval.ival)
1970                     else
1971                       if typtr = nilptr then gen2(51(*ldc*),4,0)
1972                       else
1973                         if cstptrix >= cstoccmax then error(254)
1974                         else
1975                           begin cstptrix := cstptrix + 1;
1976                             cstptr[cstptrix] := cval.valp;
```

```
1977                          if typtr = realptr then
1978                            gen2(51(*ldc*),2,cstptrix)
1979                          else
1980                            gen2(51(*ldc*),5,cstptrix)
1981                        end;
1982            varbl: case access of
1983                     drct:   if vlevel<=1 then
1984                               genlt(39(*ldo*),dplmt,typtr)
1985                             else gen2t(54(*lod*),level-vlevel,dplmt,typtr);
1986                     indrct: genlt(35(*ind*),idplmt,typtr);
1987                     inxd:   error(400)
1988                   end;
1989            expr:
1990          end;
1991          kind := expr
1992        end
1993    end (*load*) ;
1994
1995    procedure store(var fattr: attr);
1996    begin
1997      with fattr do
1998        if typtr <> nil then
1999          case access of
2000            drct:   if vlevel <= 1 then genlt(43(*sro*),dplmt,typtr)
2001                    else gen2t(56(*str*),level-vlevel,dplmt,typtr);
2002            indrct: if idplmt <> 0 then error(400)
2003                    else gen0t(26(*sto*),typtr);
2004            inxd:   error(400)
2005          end
2006    end (*store*) ;
2007
2008    procedure loadaddress;
2009    begin
2010      with gattr do
2011        if typtr <> nil then
2012          begin
2013            case kind of
2014              cst:   if string(typtr) then
2015                       if cstptrix >= cstoccmax then error(254)
2016                       else
2017                         begin cstptrix := cstptrix + 1;
2018                           cstptr[cstptrix] := cval.valp;
2019                           gen1(38(*lca*),cstptrix)
2020                         end
2021                     else error(400);
2022              varbl: case access of
2023                       drct:   if vlevel <= 1 then gen1(37(*lao*),dplmt)
2024                               else gen2(50(*lda*),level-vlevel,dplmt);
2025                       indrct: if idplmt <> 0 then
2026                                 genlt(34(*inc*),idplmt,nilptr);
2027                       inxd:   error(400)
2028                     end;
2029              expr:  error(400)
2030            end;
2031            kind := varbl; access := indrct; idplmt := 0
2032          end
2033    end (*loadaddress*) ;
2034
2035
2036    procedure genfjp(faddr: integer);
2037    begin load;
2038      if gattr.typtr <> nil then
2039        if gattr.typtr <> boolptr then error(144);
2040      if prcode then begin putic; writeln(prr,mn[33]:4,' 1':8,faddr:4) end;
```



```
2041        ic := ic + 1; mes(33)
2042      end (*genfjp*) ,
2043
2044      procedure genujpxjp(fop: oprange; fp2: integer);
2045      begin
2046       if prcode then
2047          begin putic; writeln(prr, mn[fop]:4, ' 1':8,fp2:4) end;
2048        ic := ic + 1; mes(fop)
2049      end (*genujpxjp*);
2050
2051
2052      procedure gencupent(fop: oprange; fp1,fp2: integer);
2053      begin
2054        if prcode then
2055          begin putic;
2056            writeln(prr,mn[fop]:4,fp1:4,'1':4,fp2:4)
2057          end;
2058        ic := ic + 1; mes(fop)
2059      end;
2060
2061
2062      procedure checkbnds(fsp: stp);
2063        var lmin,lmax: integer;
2064      begin
2065        if fsp <> nil then
2066          if fsp <> intptr then
2067            if fsp <> realptr then
2068              if fsp^.form <= subrange then
2069                begin
2070                  getbounds(fsp,lmin,lmax);
2071                  gen2t(45(*chk*),lmin,lmax,fsp)
2072                end
2073      end (*checkbnds*),
2074
2075
2076      procedure putlabel(labname: integer);
2077      begin if prcode then writeln(prr, 'l', labname:4)
2078      end (*putlabel*);
2079
2080      procedure statement(fsys: setofsys);
2081        label 1;
2082        var lcp: ctp; llp: lbp;
2083
2084        procedure expression(fsys: setofsys); forward;
2085
2086        procedure selector(fsys: setofsys; fcp: ctp);
2087        var lattr: attr; lcp: ctp; lsize,lmin,lmax: integer;
2088        begin
2089          with fcp^, gattr do
2090            begin typtr := idtype; kind := varbl;
2091              case klass of
2092                vars:
2093                  if vkind = actual then
2094                    begin access := drct; vlevel := vlev;
2095                      dplmt := vaddr
2096                    end
2097                  else
2098                    begin gen2t(54(*lod*),level-vlev,vaddr,nilptr);
2099                      access := indrct; idplmt := 0
2100                    end;
2101                field:
2102                  with display[disx] do
2103                    if occur = crec then
2104                      begin access := drct; vlevel := clev;
```

```
3385          store(lattr); genujpxjp(57(*ujp*),laddr); putlabel(lcix);
3386          lc := llc;
3387        end (*forstatement*) ;
3388
3389
3390        procedure withstatement;
3391          var lcp: ctp; lcnt1: disprange; llc: addrrange;
3392        begin lcnt1 := 0; llc := lc;
3393          repeat
3394            if sy = ident then
3395              begin searchid([vars,field],lcp); insymbol end
3396            else begin error(2); lcp := uvarptr end;
3397            selector(fsys + [comma,dosy],lcp);
3398            if gattr.typtr <> nil then
3399              if gattr.typtr^.form = records then
3400                if top < displimit then
3401                  begin top := top + 1; lcnt1 := lcnt1 + 1;
3402                    with display[top] do
3403                      begin fname := gattr.typtr^.fstfld;
3404                        flabel := nil
3405                      end;
3406                    if gattr.access = drct then
3407                      with display[top] do
3408                        begin occur := crec; clev := gattr.vlevel;
3409                          cdspl := gattr.dplmt
3410                        end
3411                    else
3412                      begin loadaddress;
3413                        align(nilptr,lc);
3414                        gen2t(56(*str*),0,lc,nilptr);
3415                        with display[top] do
3416                          begin occur := vrec; vdspl := lc end;
3417                        lc := lc+ptrsize;
3418                        if lc > lcmax then lcmax := lc
3419                      end
3420                  end
3421                else error(250)
3422              else error(140);
3423            test := sy <> comma;
3424            if not test then insymbol
3425          until test;
3426          if sy = dosy then insymbol else error(54);
3427          statement(fsys);
3428          top := top-lcnt1; lc := llc;
3429        end (*withstatement*) ;
3430
3431      begin (*statement*)
3432        if sy = intconst then (*label*)
3433          begin llp := display[level].flabel;
3434            while llp <> nil do
3435              with llp^ do
3436                if labval = val.ival then
3437                  begin if defined then error(165);
3438                    putlabel(labname); defined := true;
3439                    goto 1
3440                  end
3441                else llp := nextlab;
3442            error(167);
3443      1:    insymbol;                                    !
3444            if sy = colon then insymbol else error(5)
3445          end;
3446        if not (sy in fsys + [ident]) then
3447          begin error(6); skip(fsys) end;
3448        if sy in statbegsys + [ident] then
```



```
3449          begin
3450            case sy of
3451              ident:    begin searchid([vars,field,func,proc],lcp); insymbol;
3452                          if lcp^.klass = proc then call(fsys,lcp)
3453                          else assignment(lcp)
3454                        end;
3455              beginsy:  begin insymbol; compoundstatement end;
3456              gotosy:   begin insymbol; gotostatement end;
3457              ifsy:     begin insymbol; ifstatement end;
3458              casesy:   begin insymbol; casestatement end;
3459              whilesy:  begin insymbol; whilestatement end;
3460              repeatsy: begin insymbol; repeatstatement end;
3461              forsy:    begin insymbol; forstatement end;
3462              withsy:   begin insymbol; withstatement end
3463            end;
3464            if not (sy in [semicolon,endsy,elsesy,untilsy]) then
3465              begin error(6); skip(fsys) end
3466          end
3467        end (*statement*) ;
3468
3469      begin (*body*)
3470        if fprocp <> nil then entname := fprocp^.pfname
3471        else genlabel(entname);
3472        cstptrix := 0; topnew := lcaftermarkstack; topmax := lcaftermarkstack;
3473        putlabel(entname); genlabel(segsize); genlabel(stacktop);
3474        gencupent(32(*ent1*),1,segsize); gencupent(32(*ent2*),2,stacktop);
3475        if fprocp <> nil then (*copy multiple values into local cells*)
3476          begin llc1 := lcaftermarkstack;
3477            lcp := fprocp^.next;
3478            while lcp <> nil do
3479              with lcp^ do
3480                begin
3481                  align(parmptr,llc1);
3482                  if klass = vars then
3483                    if idtype <> nil then
3484                      if idtype^.form > power then
3485                        begin
3486                          if vkind = actual then
3487                            begin
3488                              gen2(50(*lda*),0,vaddr);
3489                              gen2t(54(*lod*),0,llc1,nilptr);
3490                              genl(40(*mov*),idtype^.size);
3491                            end;
3492                          llc1 := llc1 + ptrsize
3493                        end
3494                      else llc1 := llc1 + idtype^.size;
3495                  lcp := lcp^.next;
3496                end;
3497          end;
3498        lcmax := lc;
3499        repeat
3500          repeat statement(fsys + [semicolon,endsy])
3501          until not (sy in statbegsys);
3502          test := sy <> semicolon;
3503          if not test then insymbol
3504        until test;
3505        if sy = endsy then insymbol else error(13);
3506        llp := display[top].flabel; (*test for undefined labels*)
3507        while llp <> nil do
3508          with llp^ do
3509            begin
3510              if not defined then
3511                begin error(168);
3512                  writeln(output); writeln(output,' label ',labval),
```