```
MODULE DevCPP;
(**
    project         = "BlackBox"
    organization    = "www.oberon.ch"
    contributors    = "Oberon microsystems"
    version         = "System/Rsrc/About"
    copyright       = "System/Rsrc/About"
    license         = "Docu/BB-License"
    references      = "http://e-collection.library.ethz.ch/eserv/eth:39386/eth-39386-02.pdf"
    changes         = "+ -"
    issues          = "+ -"

**)

    IMPORT
        DevCPM, DevCPT, DevCPB, DevCPS;

    CONST
        anchorVarPar = TRUE;

        (* numtyp values *)
        char = 1; integer = 2; real = 4; int64 = 5; real32 = 6; real64 = 7;

        (*symbol values*)
        null = 0; times = 1; slash = 2; div = 3; mod = 4;
        and = 5; plus = 6; minus = 7; or = 8; eql = 9;
        neq = 10; lss = 11; leq = 12; gtr = 13; geq = 14;
        in = 15; is = 16; arrow = 17; dollar = 18; period = 19;
        comma = 20; colon = 21; upto = 22; rparen = 23; rbrak = 24;
        rbrace = 25; of = 26; then = 27; do = 28; to = 29;
        by = 30; not = 33;
        lparen = 40; lbrak = 41; lbrace = 42; becomes = 44;
        number = 45; nil = 46; string = 47; ident = 48; semicolon = 49;
        bar = 50; end = 51; else = 52; elsif = 53; until = 54;
        if = 55; case = 56; while = 57; repeat = 58; for = 59;
        loop = 60; with = 61; exit = 62; return = 63; array = 64;
        record = 65; pointer = 66; begin = 67; const = 68; type = 69;
        var = 70; out = 71; procedure = 72; close = 73; import = 74;
        module = 75; eof = 76;

        (* object modes *)
        Var = 1; VarPar = 2; Con = 3; Fld = 4; Typ = 5; LProc = 6; XProc = 7;
        SProc = 8; CProc = 9; IProc = 10; Mod = 11; Head = 12; TProc = 13; Attr = 20;

        (* Structure forms *)
        Undef = 0; Byte = 1; Bool = 2; Char8 = 3; Int8 = 4; Int16 = 5; Int32 = 6;
        Real32 = 7; Real64 = 8; Set = 9; String8 = 10; NilTyp = 11; NoTyp = 12;
        Pointer = 13; ProcTyp = 14; Comp = 15;
        Char16 = 16; String16 = 17; Int64 = 18;
        intSet = {Int8..Int32, Int64}; charSet = {Char8, Char16};

        (* composite structure forms *)
        Basic = 1; Array = 2; DynArr = 3; Record = 4;

        (*function number*)
        haltfn = 0; newfn = 1; incfn = 13; sysnewfn = 30;

        (* nodes classes *)
        Nvar = 0; Nvarpar = 1; Nfield = 2; Nderef = 3; Nindex = 4; Nguard = 5; Neguard = 6;
```

```
    Nconst = 7; Ntype = 8; Nproc = 9; Nupto = 10; Nmop = 11; Ndop = 12; Ncall = 13;
    Ninittd = 14; Nif = 15; NcaseIse = 16; Ncasedo = 17; Nenter = 18; Nassign = 19;
    Nifelse = 20; Ncase = 21; Nwhile = 22; Nrepeat = 23; Nloop = 24; Nexit = 25;
    Nreturn = 26; Nwith = 27; Ntrap = 28; Ncomp = 30;

    (* node subclasses *)
    super = 1;

    (* module visibility of objects *)
    internal = 0; external = 1; externalR = 2; inPar = 3; outPar = 4;

    (* procedure flags (conval.setval) *)
    hasBody = 1; isRedef = 2; slNeeded = 3; imVar = 4;

    (* attribute flags (attr.adr, struct.attribute, proc.conval.setval)*)
    newAttr = 16; absAttr = 17; limAttr = 18; empAttr = 19; extAttr = 20;

    (* case statement flags (conval.setval) *)
    useTable = 1; useTree = 2;

    (* sysflags *)
    nilBit = 1; inBit = 2; outBit = 4; newBit = 8; iidBit = 16; interface = 10; som = 20; jstr = -13;


  TYPE
    Elem = POINTER TO RECORD
        next: Elem;
        struct: DevCPT.Struct;
        obj, base: DevCPT.Object;
        pos: INTEGER;
        name: DevCPT.String
    END;


  VAR
    sym, level: BYTE;
    LoopLevel: SHORTINT;
    TDinit, lastTDinit: DevCPT.Node;
    userList: Elem;
    recList: Elem;
    hasReturn: BOOLEAN;


  PROCEDURE^ Type(VAR typ: DevCPT.Struct; VAR name: DevCPT.String);
  PROCEDURE^ Expression(VAR x: DevCPT.Node);
  PROCEDURE^ Block(VAR procdec, statseq: DevCPT.Node);

  (* forward type handling *)

  PROCEDURE IncompleteType (typ: DevCPT.Struct): BOOLEAN;
  BEGIN
    IF typ.form = Pointer THEN
        IF typ = DevCPT.sysptrtyp THEN RETURN FALSE END;
        typ := typ.BaseTyp
    END;
    RETURN (typ = DevCPT.undftyp) OR (typ.comp = Record) & (typ.BaseTyp = DevCPT.undftyp)
  END IncompleteType;

  PROCEDURE SetType (struct: DevCPT.Struct; obj: DevCPT.Object; typ: DevCPT.Struct; name: DevCPT.String);
    VAR u: Elem;
```

```
BEGIN
   IF obj # NIL THEN obj.typ := typ ELSE struct.BaseTyp := typ END;
   IF name # NIL THEN
      NEW(u); u.struct := struct; u.obj := obj; u.pos := DevCPM.errpos; u.name := name;
      u.next := userList; userList := u
   END
END SetType;

PROCEDURE CheckAlloc (VAR typ: DevCPT.Struct; dynAllowed: BOOLEAN; pos: INTEGER);
BEGIN
   typ.pvused := TRUE;
   IF typ.comp = DynArr THEN
      IF ~dynAllowed THEN DevCPM.Mark(88, pos); typ := DevCPT.undftyp END
   ELSIF typ.comp = Record THEN
      IF (typ.attribute = absAttr) OR (typ.attribute = limAttr) & (typ.mno # 0) THEN
         DevCPM.Mark(193, pos); typ := DevCPT.undftyp
      END
   END
END CheckAlloc;

PROCEDURE CheckRecursiveType (outer, inner: DevCPT.Struct; pos: INTEGER);
   VAR fld: DevCPT.Object;
BEGIN
   IF outer = inner THEN DevCPM.Mark(58, pos)
   ELSIF inner.comp IN {Array, DynArr} THEN CheckRecursiveType(outer, inner.BaseTyp, pos)
   ELSIF inner.comp = Record THEN
      fld := inner.link;
      WHILE (fld # NIL) & (fld.mode = Fld) DO
         CheckRecursiveType(outer, fld.typ, pos);
         fld := fld.link
      END;
      IF inner.BaseTyp # NIL THEN CheckRecursiveType(outer, inner.BaseTyp, pos) END
   END
END CheckRecursiveType;

PROCEDURE FixType (struct: DevCPT.Struct; obj: DevCPT.Object; typ: DevCPT.Struct; pos: INTEGER);
(* fix forward reference *)
   VAR t: DevCPT.Struct; f, bf: DevCPT.Object; i: SHORTINT;
BEGIN
   IF obj # NIL THEN
      IF obj.mode = Var THEN    (* variable type *)
         IF struct # NIL THEN    (* receiver type *)
            IF (typ.form # Pointer) OR (typ.BaseTyp # struct) THEN DevCPM.Mark(180, pos) END;
         ELSE CheckAlloc(typ, obj.mnolev > level, pos)    (* TRUE for parameters *)
         END
      ELSIF obj.mode = VarPar THEN    (* varpar type *)
         IF struct # NIL THEN    (* varpar receiver type *)
            IF typ # struct THEN DevCPM.Mark(180, pos) END
         END
      ELSIF obj.mode = Fld THEN    (* field type *)
         CheckAlloc(typ, FALSE, pos);
         CheckRecursiveType(struct, typ, pos)
      ELSIF obj.mode = TProc THEN    (* proc return type *)
         IF typ.form = Comp THEN typ := DevCPT.undftyp; DevCPM.Mark(54, pos) END
      ELSIF obj.mode = Typ THEN    (* alias type *)
         IF typ.form IN {Byte..Set, Char16, Int64} THEN    (* make alias structure *)
            t := DevCPT.NewStr(typ.form, Basic); i := t.ref;
            t^ := typ^; t.ref := i; t.strobj := obj; t.mno := 0;
            t.BaseTyp := typ; typ := t
         END;
```

```
            IF obj.vis # internal THEN
               IF typ.comp = Record THEN typ.exp := TRUE
               ELSIF typ.form = Pointer THEN typ.BaseTyp.exp := TRUE
               END
            END
         ELSE HALT(100)
         END;
         obj.typ := typ
      ELSE
         IF struct.form = Pointer THEN    (* pointer base type *)
            IF typ.comp = Record THEN DevCPM.PropagateRecPtrSysFlag(typ.sysflag, struct.sysflag)
            ELSIF typ.comp IN {Array, DynArr} THEN DevCPM.PropagateArrPtrSysFlag(typ.sysflag, struct.sysflag)
            ELSE typ := DevCPT.undftyp; DevCPM.Mark(57, pos)
            END;
            struct.untagged := struct.sysflag > 0;
            IF (struct.strobj # NIL) & (struct.strobj.vis # internal) THEN typ.exp := TRUE END;
         ELSIF struct.comp = Array THEN    (* array base type *)
            CheckAlloc(typ, FALSE, pos);
            CheckRecursiveType(struct, typ, pos)
         ELSIF struct.comp = DynArr THEN    (* array base type *)
            CheckAlloc(typ, TRUE, pos);
            CheckRecursiveType(struct, typ, pos)
         ELSIF struct.comp = Record THEN    (* record base type *)
            IF typ.form = Pointer THEN typ := typ.BaseTyp END;
            typ.pvused := TRUE; struct.extlev := SHORT(SHORT(typ.extlev + 1));
            DevCPM.PropagateRecordSysFlag(typ.sysflag, struct.sysflag);
            IF (typ.attribute = 0) OR (typ.attribute = limAttr) & (typ.mno # 0) THEN DevCPM.Mark(181, pos)
            ELSIF (struct.attribute = absAttr) & (typ.attribute # absAttr) THEN DevCPM.Mark(191, pos)
            ELSIF (typ.attribute = limAttr) & (struct.attribute # limAttr) THEN DevCPM.Mark(197, pos)
            END;
            f := struct.link;
            WHILE f # NIL DO    (* check for field name conflicts *)
               DevCPT.FindField(f.name, typ, bf);
               IF bf # NIL THEN DevCPM.Mark(1, pos) END;
               f := f.link
            END;
            CheckRecursiveType(struct, typ, pos);
            struct.untagged := struct.sysflag > 0;
         ELSIF struct.form = ProcTyp THEN    (* proc type return type *)
            IF typ.form = Comp THEN typ := DevCPT.undftyp; DevCPM.Mark(54, pos) END;
         ELSE HALT(100)
         END;
         struct.BaseTyp := typ
      END
   END FixType;

   PROCEDURE CheckForwardTypes;
      VAR u, next: Elem; progress: BOOLEAN;
   BEGIN
      u := userList; userList := NIL;
      WHILE u # NIL DO
         next := u.next; DevCPS.name := u.name$; DevCPT.Find(DevCPS.name, u.base);
         IF u.base = NIL THEN DevCPM.Mark(0, u.pos)
         ELSIF u.base.mode # Typ THEN DevCPM.Mark(72, u.pos)
         ELSE u.next := userList; userList := u    (* reinsert *)
         END;
         u := next
      END;
      REPEAT    (* iteration for multy level alias *)
         u := userList; userList := NIL; progress := FALSE;
```

```
      WHILE u # NIL DO
        next := u.next;
        IF IncompleteType(u.base.typ) THEN
          u.next := userList; userList := u    (* reinsert *)
        ELSE
          progress := TRUE;
          FixType(u.struct, u.obj, u.base.typ, u.pos)
        END;
        u := next
      END
    UNTIL (userList = NIL) OR ~progress;
    u := userList;    (* remaining type relations are cyclic *)
    WHILE u # NIL DO
      IF (u.obj = NIL) OR (u.obj.mode = Typ) THEN DevCPM.Mark(58, u.pos) END;
      u := u.next
    END;
  END CheckForwardTypes;

  PROCEDURE CheckUnimpl (m: DevCPT.Object; typ: DevCPT.Struct; pos: INTEGER);
    VAR obj: DevCPT.Object;
  BEGIN
    IF m # NIL THEN
      IF (m.mode = TProc) & (absAttr IN m.conval.setval) THEN
        DevCPT.FindField(m.name^, typ, obj);
        IF (obj = NIL) OR (obj.mode # TProc) OR (absAttr IN obj.conval.setval) THEN
          DevCPM.Mark(192, pos);
          DevCPM.LogWLn; DevCPM.LogWStr("  ");
          IF typ.strobj # NIL THEN
            DevCPM.LogWPar("#Dev:NotImplementedIn", m.name, typ.strobj.name)
          ELSE
            DevCPM.LogWPar("#Dev:NotImplemented", m.name, "")
          END
        END
      END;
      CheckUnimpl(m.left, typ, pos);
      CheckUnimpl(m.right, typ, pos)
    END
  END CheckUnimpl;

  PROCEDURE CheckRecords (rec: Elem);
    VAR b: DevCPT.Struct;
  BEGIN
    WHILE rec # NIL DO    (* check for unimplemented methods in base type *)
      b := rec.struct.BaseTyp;
      WHILE (b # NIL) & (b # DevCPT.undftyp) DO
        CheckUnimpl(b.link, rec.struct, rec.pos);
        b := b.BaseTyp
      END;
      rec := rec.next
    END
  END CheckRecords;


  PROCEDURE err(n: SHORTINT);
  BEGIN DevCPM.err(n)
  END err;

  PROCEDURE CheckSym(s: SHORTINT);
  BEGIN
    IF sym = s THEN DevCPS.Get(sym) ELSE DevCPM.err(s) END
```

```
END CheckSym;

PROCEDURE qualident(VAR id: DevCPT.Object);
   VAR obj: DevCPT.Object; lev: BYTE;
BEGIN (*sym = ident*)
   DevCPT.Find(DevCPS.name, obj); DevCPS.Get(sym);
   IF (sym = period) & (obj # NIL) & (obj.mode = Mod) THEN
      DevCPS.Get(sym);
      IF sym = ident THEN
         DevCPT.FindImport(DevCPS.name, obj, obj); DevCPS.Get(sym)
      ELSE err(ident); obj := NIL
      END
   END ;
   IF obj = NIL THEN err(0);
      obj := DevCPT.NewObj(); obj.mode := Var; obj.typ := DevCPT.undftyp; obj.adr := 0
   ELSE lev := obj.mnolev;
      IF (obj.mode IN {Var, VarPar}) & (lev # level) THEN
         obj.leaf := FALSE;
         IF lev > 0 THEN DevCPB.StaticLink(SHORT(SHORT(level-lev)), TRUE) END     (* !!! *)
      END
   END ;
   id := obj
END qualident;

PROCEDURE ConstExpression(VAR x: DevCPT.Node);
BEGIN Expression(x);
   IF x.class # Nconst THEN
      err(50); x := DevCPB.NewIntConst(1)
   END
END ConstExpression;

PROCEDURE CheckMark(obj: DevCPT.Object);     (* !!! *)
   VAR n: INTEGER; mod: ARRAY 256 OF DevCPT.String;
BEGIN DevCPS.Get(sym);
   IF (sym = times) OR (sym = minus) THEN
      IF (level > 0) OR ~(obj.mode IN {Var, Fld, TProc}) & (sym = minus) THEN err(41) END ;
      IF sym = times THEN obj.vis := external ELSE obj.vis := externalR END ;
      DevCPS.Get(sym)
   ELSE obj.vis := internal
   END;
   IF (obj.mode IN {TProc, LProc, XProc, CProc, Var, Typ, Con, Fld}) & (sym = lbrak) THEN
      DevCPS.Get(sym);
      IF (sym = number) & (DevCPS.numtyp = char) THEN
         NEW(DevCPS.str, 2); DevCPS.str[0] := SHORT(CHR(DevCPS.intval)); DevCPS.str[1] := 0X; sym := string
      END;
      IF sym = string THEN
         IF DevCPS.str^ # "" THEN obj.entry := DevCPS.str END;
         DevCPS.Get(sym); n := 0;
         IF (sym = comma) & (obj.mode IN {LProc, XProc, CProc, Var, Con}) THEN
            DevCPS.Get(sym);
            IF (sym = number) & (DevCPS.numtyp = char) THEN
               NEW(DevCPS.str, 2); DevCPS.str[0] := SHORT(CHR(DevCPS.intval)); DevCPS.str[1] := 0X; sym := string
            END;
            IF sym = string THEN
               obj.library := obj.entry; obj.entry := NIL;
               IF DevCPS.str^ # "" THEN obj.entry := DevCPS.str END;
               DevCPS.Get(sym);
            ELSE err(string)
            END
         END;
```

```
            WHILE sym = comma DO
               DevCPS.Get(sym);
               IF (sym = number) & (DevCPS.numtyp = char) THEN
                  NEW(DevCPS.str, 2); DevCPS.str[0] := SHORT(CHR(DevCPS.intval)); DevCPS.str[1] := 0X; sym := string
               END;
               IF sym = string THEN
                  IF n < LEN(mod) THEN mod[n] := DevCPS.str; INC(n)
                  ELSE err(235)
                  END;
                  DevCPS.Get(sym)
               ELSE err(string)
               END
            END;
            IF n > 0 THEN
               NEW(obj.modifiers, n);
               WHILE n > 0 DO DEC(n); obj.modifiers[n] := mod[n] END
            END
         ELSE err(string)
         END;
         CheckSym(rbrak);
         IF DevCPM.options * {DevCPM.interface, DevCPM.java} = {} THEN err(225) END
      END
   END CheckMark;

   PROCEDURE CheckSysFlag (VAR sysflag: SHORTINT;
                           GetSF: PROCEDURE(IN id: ARRAY OF SHORTCHAR; num: SHORTINT; VAR flag:
SHORTINT));
      VAR x: DevCPT.Object; i: SHORTINT;
   BEGIN
      sysflag := 0;
      IF sym = lbrak THEN
         DevCPS.Get(sym);
         WHILE (sym = number) OR (sym = ident) OR (sym = string) DO
            IF sym = number THEN
               IF DevCPS.numtyp = integer THEN
                  i := SHORT(DevCPS.intval); GetSF("", i, sysflag)
               ELSE err(225)
               END
            ELSIF sym = ident THEN
               DevCPT.Find(DevCPS.name, x);
               IF (x # NIL) & (x.mode = Con) & (x.typ.form IN {Int8, Int16, Int32}) THEN
                  i := SHORT(x.conval.intval); GetSF("", i, sysflag)
               ELSE
                  GetSF(DevCPS.name, 0, sysflag)
               END
            ELSE
               GetSF(DevCPS.str, 0, sysflag)
            END;
            DevCPS.Get(sym);
            IF (sym = comma) OR (sym = plus) THEN DevCPS.Get(sym) END
         END;
         CheckSym(rbrak)
      END
   END CheckSysFlag;

   PROCEDURE Receiver(VAR mode, vis: BYTE; VAR name: DevCPT.Name; VAR typ, rec: DevCPT.Struct);
      VAR tname: DevCPT.String;
   BEGIN typ := DevCPT.undftyp; rec := NIL; vis := 0;
      IF sym = var THEN DevCPS.Get(sym); mode := VarPar;
      ELSIF sym = in THEN DevCPS.Get(sym); mode := VarPar; vis := inPar     (* ??? *)
```

```
    ELSE mode := Var
    END ;
    name := DevCPS.name; CheckSym(ident); CheckSym(colon);
    IF sym # ident THEN err(ident) END;
    Type(typ, tname);
    IF tname = NIL THEN
       IF typ.form = Pointer THEN  rec := typ.BaseTyp ELSE rec := typ END;
       IF ~((mode = Var) & (typ.form = Pointer) & (rec.comp = Record) OR
          (mode = VarPar) & (typ.comp = Record)) THEN err(70); rec := NIL END;
       IF (rec # NIL) & (rec.mno # level) THEN err(72); rec := NIL END
    ELSE err(0)
    END;
    CheckSym(rparen);
    IF rec = NIL THEN rec := DevCPT.NewStr(Comp, Record); rec.BaseTyp := NIL END
END Receiver;

PROCEDURE FormalParameters(
    VAR firstPar: DevCPT.Object; VAR resTyp: DevCPT.Struct; VAR name: DevCPT.String
);
    VAR mode, vis: BYTE; sys: SHORTINT;
        par, first, last, newPar, iidPar: DevCPT.Object; typ: DevCPT.Struct;
BEGIN
    first := NIL; last := firstPar;
    newPar := NIL; iidPar := NIL;
    IF (sym = ident) OR (sym = var) OR (sym = in) OR (sym = out) THEN
       LOOP
          sys := 0; vis := 0;
          IF sym = var THEN DevCPS.Get(sym); mode := VarPar
          ELSIF sym = in THEN DevCPS.Get(sym); mode := VarPar; vis := inPar
          ELSIF sym = out THEN DevCPS.Get(sym); mode := VarPar; vis := outPar
          ELSE mode := Var
          END ;
          IF mode = VarPar THEN CheckSysFlag(sys, DevCPM.GetVarParSysFlag) END;
          IF ODD(sys DIV inBit) THEN vis := inPar
          ELSIF ODD(sys DIV outBit) THEN vis := outPar
          END;
          IF ODD(sys DIV newBit) & (vis # outPar) THEN err(225)
          ELSIF ODD(sys DIV iidBit) & (vis # inPar) THEN err(225)
          END;
          LOOP
             IF sym = ident THEN
                DevCPT.Insert(DevCPS.name, par); DevCPS.Get(sym);
                par.mode := mode; par.link := NIL; par.vis := vis; par.sysflag := SHORT(sys);
                IF first = NIL THEN first := par END ;
                IF firstPar = NIL THEN firstPar := par ELSE last.link := par END ;
                last := par
             ELSE err(ident)
             END;
             IF sym = comma THEN DevCPS.Get(sym)
             ELSIF sym = ident THEN err(comma)
             ELSIF sym = var THEN err(comma); DevCPS.Get(sym)
             ELSE EXIT
             END
          END ;
          CheckSym(colon); Type(typ, name);
          IF mode # VarPar THEN CheckAlloc(typ, TRUE, DevCPM.errpos) END;
          (* IN only allowed for records and arrays *)
          IF (mode = VarPar) & (vis = inPar) & (typ.form # Undef) & (typ.form # Comp) & (typ.sysflag = 0) THEN err(177)
          END;
          (* typ.pbused is set when parameter type name is parsed *)
```

```
            WHILE first # NIL DO
               SetType (NIL, first, typ, name);
               IF DevCPM.com IN DevCPM.options THEN
                  IF ODD(sys DIV newBit) THEN
                     IF (newPar # NIL) OR (typ.form # Pointer) OR (typ.sysflag # interface) THEN err(168) END;
                     newPar := first
                  ELSIF ODD(sys DIV iidBit) THEN
                     IF (iidPar # NIL) OR (typ # DevCPT.guidtyp) THEN err(168) END;
                     iidPar := first
                  END
               END;
               first := first.link
            END;
            IF sym = semicolon THEN DevCPS.Get(sym)
            ELSIF sym = ident THEN err(semicolon)
            ELSE EXIT
            END
         END
      END;
      CheckSym(rparen);
      IF (newPar = NIL) # (iidPar = NIL) THEN err(168) END;
      name := NIL;
      IF sym = colon THEN
         DevCPS.Get(sym);
         Type(resTyp, name);
         IF resTyp.form = Comp THEN resTyp := DevCPT.undftyp; err(54) END
      ELSE resTyp := DevCPT.notyp
      END
   END FormalParameters;

   PROCEDURE CheckOverwrite (proc, base: DevCPT.Object; rec: DevCPT.Struct);
      VAR o, bo: DevCPT.Object;
   BEGIN
      IF base # NIL THEN
         IF base.conval.setval * {absAttr, empAttr, extAttr} = {} THEN err(182) END;
         IF (proc.link.mode # base.link.mode) OR (proc.link.vis # base.link.vis)
            OR ~DevCPT.Extends(proc.link.typ, base.link.typ) THEN err(115) END;
         o := proc.link; bo := base.link;
         WHILE (o # NIL) & (bo # NIL) DO
            IF (bo.sysflag # 0) & (o.sysflag = 0) THEN    (* propagate sysflags *)
               o.sysflag := bo.sysflag
            END;
            o := o.link; bo := bo.link
         END;
         DevCPB.CheckParameters(proc.link.link, base.link.link, FALSE);
         IF ~DevCPT.Extends(proc.typ, base.typ) THEN err(117) END;
         IF (base.vis # proc.vis) & ((proc.vis # internal) OR rec.exp) THEN err(183) END;
         INCL(proc.conval.setval, isRedef)
      END;
   END CheckOverwrite;

   PROCEDURE GetAttributes (proc, base: DevCPT.Object; owner: DevCPT.Struct);    (* read method attributes *)
      VAR attr, battr: SET; o: DevCPT.Object;
   BEGIN
      attr := {};
      IF sym = comma THEN    (* read attributes *)
         DevCPS.Get(sym);
         IF sym = ident THEN
            DevCPT.Find(DevCPS.name, o);
            IF (o # NIL) & (o.mode = SProc) & (o.adr = newfn) THEN
```

```
                IF ~(DevCPM.oberon IN DevCPM.options) THEN INCL(attr, newAttr) ELSE err(178) END;
                DevCPS.Get(sym);
                IF sym = comma THEN
                  DevCPS.Get(sym);
                  IF sym = ident THEN DevCPT.Find(DevCPS.name, o) ELSE o := NIL; err(ident) END
                ELSE o := NIL
                END
              END;
              IF o # NIL THEN
                IF (o.mode # Attr) OR (o.adr = limAttr) OR (DevCPM.oberon IN DevCPM.options) THEN err(178)
                ELSE INCL(attr, o.adr)
                END;
                DevCPS.Get(sym)
              END
            ELSE err(ident)
            END
          END;
          IF (base = NIL) & ~(newAttr IN attr) THEN err(185); INCL(attr, newAttr)
          ELSIF (base # NIL) & (newAttr IN attr) THEN err(186)
          END;
          IF absAttr IN attr THEN
            IF owner.attribute # absAttr THEN err(190) END;
            IF (proc.vis = internal) & owner.exp THEN err(179) END
          END;
          IF (owner.attribute = 0) OR (owner.attribute = limAttr) THEN
            IF (empAttr IN attr) & (newAttr IN attr) THEN err(187)
(*
            ELSIF extAttr IN attr THEN err(188)
*)
            END
          END;
          IF base # NIL THEN
            battr := base.conval.setval;
            IF empAttr IN battr THEN
              IF absAttr IN attr THEN err(189) END
            ELSIF ~(absAttr IN battr) THEN
              IF (absAttr IN attr) OR (empAttr IN attr) THEN err(189) END
            END
          END;
          IF empAttr IN attr THEN
            IF proc.typ # DevCPT.notyp THEN err(195)
            ELSE
              o := proc.link; WHILE (o # NIL) & (o.vis # outPar) DO o := o.link END;
              IF o # NIL THEN err(195) END
            END
          END;
          IF (owner.sysflag = interface) & ~(absAttr IN attr) THEN err(162) END;
          proc.conval.setval := attr
        END GetAttributes;

        PROCEDURE RecordType(VAR typ: DevCPT.Struct; attr: DevCPT.Object);
          VAR fld, first, last: DevCPT.Object; r: Elem; ftyp: DevCPT.Struct; name: DevCPT.String;
        BEGIN typ := DevCPT.NewStr(Comp, Record); typ.BaseTyp := NIL;
          CheckSysFlag(typ.sysflag, DevCPM.GetRecordSysFlag);
          IF attr # NIL THEN
            IF ~(DevCPM.oberon IN DevCPM.options) & (attr.adr # empAttr) THEN typ.attribute := SHORT(SHORT(attr.adr))
            ELSE err(178)
            END
          END;
          IF typ.sysflag = interface THEN
```

```
            IF (DevCPS.str # NIL) & (DevCPS.str[0] = "{") THEN typ.ext := DevCPS.str END;
            IF typ.attribute # absAttr THEN err(163) END;
            IF sym # lparen THEN err(160) END
        END;
        IF sym = lparen THEN
            DevCPS.Get(sym); (*record extension*)
            IF sym = ident THEN
                Type(ftyp, name);
                IF ftyp.form = Pointer THEN ftyp := ftyp.BaseTyp END;
                SetType(typ, NIL, ftyp, name);
                IF (ftyp.comp = Record) & (ftyp # DevCPT.anytyp) THEN
                    ftyp.pvused := TRUE; typ.extlev := SHORT(SHORT(ftyp.extlev + 1));
                    DevCPM.PropagateRecordSysFlag(ftyp.sysflag, typ.sysflag);
                    IF (ftyp.attribute = 0) OR (ftyp.attribute = limAttr) & (ftyp.mno # 0) THEN err(181)
                    ELSIF (typ.attribute = absAttr) & (ftyp.attribute # absAttr) & ~(DevCPM.java IN DevCPM.options) THEN err(191)
                    ELSIF (ftyp.attribute = limAttr) & (typ.attribute # limAttr) THEN err(197)
                    END
                ELSIF ftyp # DevCPT.undftyp THEN err(53)
                END
            ELSE err(ident)
            END ;
            IF typ.attribute # absAttr THEN    (* save typ for unimplemented method check *)
                NEW(r); r.struct := typ; r.pos := DevCPM.errpos; r.next := recList; recList := r
            END;
            CheckSym(rparen)
        END;
(*
        DevCPT.OpenScope(0, NIL);
*)
        first := NIL; last := NIL;
        LOOP
            IF sym = ident THEN
                LOOP
                    IF sym = ident THEN
                        IF (typ.BaseTyp # NIL) & (typ.BaseTyp # DevCPT.undftyp) THEN
                            DevCPT.FindBaseField(DevCPS.name, typ, fld);
                            IF fld # NIL THEN err(1) END
                        END ;
                        DevCPT.InsertField(DevCPS.name, typ, fld);
                        fld.mode := Fld; fld.link := NIL; fld.typ := DevCPT.undftyp;
                        CheckMark(fld);
                        IF first = NIL THEN first := fld END ;
                        IF last = NIL THEN typ.link := fld ELSE last.link := fld END ;
                        last := fld
                    ELSE err(ident)
                    END ;
                    IF sym = comma THEN DevCPS.Get(sym)
                    ELSIF sym = ident THEN err(comma)
                    ELSE EXIT
                    END
                END ;
                CheckSym(colon); Type(ftyp, name);
                CheckAlloc(ftyp, FALSE, DevCPM.errpos);
                WHILE first # NIL DO
                    SetType(typ, first, ftyp, name); first := first.link
                END;
                IF typ.sysflag = interface THEN err(161) END
            END;
            IF sym = semicolon THEN DevCPS.Get(sym)
            ELSIF sym = ident THEN err(semicolon)
```

```
        ELSE EXIT
        END
      END;
(*
      IF typ.link # NIL THEN ASSERT(typ.link = DevCPT.topScope.right) END;
      typ.link := DevCPT.topScope.right; DevCPT.CloseScope;
*)
      typ.untagged := typ.sysflag > 0;
      DevCPB.Inittd(TDinit, lastTDinit, typ); CheckSym(end)
    END RecordType;

    PROCEDURE ArrayType(VAR typ: DevCPT.Struct);
      VAR x: DevCPT.Node; n: INTEGER; sysflag: SHORTINT; name: DevCPT.String;
    BEGIN CheckSysFlag(sysflag, DevCPM.GetArraySysFlag);
      IF sym = of THEN    (*dynamic array*)
        typ := DevCPT.NewStr(Comp, DynArr); typ.mno := 0; typ.sysflag := sysflag;
        DevCPS.Get(sym); Type(typ.BaseTyp, name); SetType(typ, NIL, typ.BaseTyp, name);
        CheckAlloc(typ.BaseTyp, TRUE, DevCPM.errpos);
        IF typ.BaseTyp.comp = DynArr THEN typ.n := typ.BaseTyp.n + 1 ELSE typ.n := 0 END
      ELSE
        typ := DevCPT.NewStr(Comp, Array); typ.sysflag := sysflag; ConstExpression(x);
        IF x.typ.form IN {Int8, Int16, Int32} THEN n := x.conval.intval;
          IF (n <= 0) OR (n > DevCPM.MaxIndex) THEN err(63); n := 1 END
        ELSE err(42); n := 1
        END ;
        typ.n := n;
        IF sym = of THEN
          DevCPS.Get(sym); Type(typ.BaseTyp, name); SetType(typ, NIL, typ.BaseTyp, name);
          CheckAlloc(typ.BaseTyp, FALSE, DevCPM.errpos)
        ELSIF sym = comma THEN
          DevCPS.Get(sym);
          IF sym # of THEN ArrayType(typ.BaseTyp) END
        ELSE err(35)
        END
      END;
      typ.untagged := typ.sysflag > 0
    END ArrayType;

    PROCEDURE PointerType(VAR typ: DevCPT.Struct);
      VAR name: DevCPT.String;
    BEGIN typ := DevCPT.NewStr(Pointer, Basic); CheckSysFlag(typ.sysflag, DevCPM.GetPointerSysFlag);
      CheckSym(to);
      Type(typ.BaseTyp, name);
      SetType(typ, NIL, typ.BaseTyp, name);
      IF (typ.BaseTyp # DevCPT.undftyp) & (typ.BaseTyp.comp = Basic) THEN
        typ.BaseTyp := DevCPT.undftyp; err(57)
      END;
      IF typ.BaseTyp.comp = Record THEN DevCPM.PropagateRecPtrSysFlag(typ.BaseTyp.sysflag, typ.sysflag)
      ELSIF typ.BaseTyp.comp IN {Array, DynArr} THEN DevCPM.PropagateArrPtrSysFlag(typ.BaseTyp.sysflag, typ.sysflag)
      END;
      typ.untagged := typ.sysflag > 0
    END PointerType;

    PROCEDURE Type (VAR typ: DevCPT.Struct; VAR name: DevCPT.String);    (* name # NIL => forward reference *)
      VAR id: DevCPT.Object; tname: DevCPT.String;
    BEGIN
      typ := DevCPT.undftyp; name := NIL;
      IF sym < lparen THEN err(12);
        REPEAT DevCPS.Get(sym) UNTIL sym >= lparen
      END ;
```

```
    IF sym = ident THEN
      DevCPT.Find(DevCPS.name, id);
      IF (id = NIL) OR (id.mode = -1) OR (id.mode = Typ) & IncompleteType(id.typ) THEN    (* forward type definition *)
        name := DevCPT.NewName(DevCPS.name); DevCPS.Get(sym);
        IF (id = NIL) & (sym = period) THEN    (* missing module *)
          err(0); DevCPS.Get(sym); name := NIL;
          IF sym = ident THEN DevCPS.Get(sym) END
        ELSIF sym = record THEN    (* wrong attribute *)
          err(178); DevCPS.Get(sym); name := NIL; RecordType(typ, NIL)
        END
      ELSE
        qualident(id);
        IF id.mode = Typ THEN
          IF ~(DevCPM.oberon IN DevCPM.options)
            & ((id.typ = DevCPT.lreal64typ) OR (id.typ = DevCPT.lint64typ) OR (id.typ = DevCPT.lchar16typ)) THEN
            err(198)
          END;
          typ := id.typ
        ELSIF id.mode = Attr THEN
          IF sym = record THEN
            DevCPS.Get(sym); RecordType(typ, id)
          ELSE err(12)
          END
        ELSE err(52)
        END
      END
    ELSIF sym = array THEN
      DevCPS.Get(sym); ArrayType(typ)
    ELSIF sym = record THEN
      DevCPS.Get(sym); RecordType(typ, NIL)
    ELSIF sym = pointer THEN
      DevCPS.Get(sym); PointerType(typ)
    ELSIF sym = procedure THEN
      DevCPS.Get(sym); typ := DevCPT.NewStr(ProcTyp, Basic);
      CheckSysFlag(typ.sysflag, DevCPM.GetProcTypSysFlag);
      typ.untagged := typ.sysflag > 0;
      IF sym = lparen THEN
        DevCPS.Get(sym); DevCPT.OpenScope(level, NIL);
        FormalParameters(typ.link, typ.BaseTyp, tname); SetType(typ, NIL, typ.BaseTyp, tname); DevCPT.CloseScope
      ELSE typ.BaseTyp := DevCPT.notyp; typ.link := NIL
      END
    ELSE err(12)
    END ;
    LOOP
      IF (sym >= semicolon) & (sym <= else) OR (sym = rparen) OR (sym = eof)
        OR (sym = number) OR (sym = comma) OR (sym = string) THEN EXIT END;
      err(15); IF sym = ident THEN EXIT END;
      DevCPS.Get(sym)
    END
  END Type;

  PROCEDURE ActualParameters(VAR aparlist: DevCPT.Node; fpar: DevCPT.Object; VAR pre, lastp: DevCPT.Node);
    VAR apar, last, newPar, iidPar, n: DevCPT.Node;
  BEGIN
    aparlist := NIL; last := NIL;
    IF sym # rparen THEN
      newPar := NIL; iidPar := NIL;
      LOOP Expression(apar);
        IF fpar # NIL THEN
          IF (apar.typ.form = Pointer) & (fpar.typ.form = Comp) THEN DevCPB.DeRef(apar) END;
```

```
            DevCPB.Param(apar, fpar);
            IF (fpar.mode = Var) OR (fpar.vis = inPar) THEN DevCPB.CheckBuffering(apar, NIL, fpar, pre, lastp) END;
            DevCPB.Link(aparlist, last, apar);
            IF ODD(fpar.sysflag DIV newBit) THEN newPar := apar
            ELSIF ODD(fpar.sysflag DIV iidBit) THEN iidPar := apar
            END;
            IF (newPar # NIL) & (iidPar # NIL) THEN DevCPB.CheckNewParamPair(newPar, iidPar) END;
            IF anchorVarPar & (fpar.mode = VarPar) & ~(DevCPM.java IN DevCPM.options)
               OR (DevCPM.allSysVal IN DevCPM.options)    (* source output: avoid double evaluation *)
                  & ((fpar.mode = VarPar) & (fpar.typ.comp = Record) & ~fpar.typ.untagged
                     OR (fpar.typ.comp = DynArr) & ~fpar.typ.untagged) THEN
               n := apar;
               WHILE n.class IN {Nfield, Nindex, Nguard} DO n := n.left END;
               IF (n.class = Nderef) & (n.subcl = 0) THEN
                  IF n.left.class = Nguard THEN n := n.left END;
                  DevCPB.CheckVarParBuffering(n.left, pre, lastp)
               END
            END;
            fpar := fpar.link
         ELSE err(64)
         END;
         IF sym = comma THEN DevCPS.Get(sym)
         ELSIF (lparen <= sym) & (sym <= ident) THEN err(comma)
         ELSE EXIT
         END
      END
   END;
   IF fpar # NIL THEN err(65) END
END ActualParameters;

PROCEDURE selector(VAR x: DevCPT.Node);
   VAR obj, proc, p, fpar: DevCPT.Object; y, apar, pre, lastp: DevCPT.Node; typ: DevCPT.Struct; name: DevCPT.Name;
BEGIN
   LOOP
      IF sym = lbrak THEN DevCPS.Get(sym);
         LOOP
            IF (x.typ # NIL) & (x.typ.form = Pointer) THEN DevCPB.DeRef(x) END ;
            Expression(y); DevCPB.Index(x, y);
            IF sym = comma THEN DevCPS.Get(sym) ELSE EXIT END
         END ;
         CheckSym(rbrak)
      ELSIF sym = period THEN DevCPS.Get(sym);
         IF sym = ident THEN name := DevCPS.name; DevCPS.Get(sym);
            IF x.typ # NIL THEN
               IF x.typ.form = Pointer THEN DevCPB.DeRef(x) END ;
               IF x.typ.comp = Record THEN
                  typ := x.typ; DevCPT.FindField(name, typ, obj); DevCPB.Field(x, obj);
                  IF (obj # NIL) & (obj.mode = TProc) THEN
                     IF sym = arrow THEN  (* super call *) DevCPS.Get(sym);
                        y := x.left;
                        IF y.class = Nderef THEN y := y.left END ;    (* y = record variable *)
                        IF y.obj # NIL THEN
                           proc := DevCPT.topScope;    (* find innermost scope which owner is a TProc *)
                           WHILE (proc.link # NIL) & (proc.link.mode # TProc) DO proc := proc.left END ;
                           IF (proc.link = NIL) OR (proc.link.link # y.obj) (* OR (proc.link.name^ # name) *) THEN err(75)
                           END ;
                           typ := y.obj.typ;
                           IF typ.form = Pointer THEN typ := typ.BaseTyp END ;
                           DevCPT.FindBaseField(x.obj.name^, typ, p);
                           IF p # NIL THEN
```

```
                        x.subcl := super; x.typ := p.typ;    (* correct result type *)
                        IF p.conval.setval * {absAttr, empAttr} # {} THEN err(194) END;
                        IF (p.vis = externalR) & (p.mnolev < 0) & (proc.link.name^ # name) THEN err(196) END;
                     ELSE err(74)
                     END
                  ELSE err(75)
                  END
               ELSE
                  proc := obj;
                  IF (x.left.readonly) & (proc.link.mode = VarPar) & (proc.link.vis = 0)  THEN err(76) END;
                  WHILE (proc.mnolev >= 0) & ~(newAttr IN proc.conval.setval) & (typ.BaseTyp # NIL) DO
                     (* find base method *)
                     typ := typ.BaseTyp; DevCPT.FindField(name, typ, proc);
                  END;
                  IF (proc.vis = externalR) & (proc.mnolev < 0) THEN err(196) END;
               END ;
               IF (obj.typ # DevCPT.notyp) & (sym # lparen) THEN err(lparen) END
            END
         ELSE err(53)
         END
      ELSE err(52)
      END
   ELSE err(ident)
   END
ELSIF sym = arrow THEN DevCPS.Get(sym); DevCPB.DeRef(x)
ELSIF sym = dollar THEN
   IF x.typ.form = Pointer THEN DevCPB.DeRef(x) END;
   DevCPS.Get(sym); DevCPB.StrDeref(x)
ELSIF sym = lparen THEN
   IF (x.obj # NIL) & (x.obj.mode IN {XProc, LProc, CProc, TProc}) THEN typ := x.obj.typ
   ELSIF x.typ.form = ProcTyp THEN typ := x.typ.BaseTyp
   ELSIF x.class = Nproc THEN EXIT    (* standard procedure *)
   ELSE typ := NIL
   END;
   IF typ # DevCPT.notyp THEN
      DevCPS.Get(sym);
      IF typ = NIL THEN    (* type guard *)
         IF sym = ident THEN
            qualident(obj);
            IF obj.mode = Typ THEN DevCPB.TypTest(x, obj, TRUE)
            ELSE err(52)
            END
         ELSE err(ident)
         END
      ELSE    (* function call *)
         pre := NIL; lastp := NIL;
         DevCPB.PrepCall(x, fpar);
         IF (x.obj # NIL) & (x.obj.mode = TProc) THEN DevCPB.CheckBuffering(x.left, NIL, x.obj.link, pre, lastp)
         END;
         ActualParameters(apar, fpar, pre, lastp);
         DevCPB.Call(x, apar, fpar);
         IF pre # NIL THEN DevCPB.Construct(Ncomp, pre, x); pre.typ := x.typ; x := pre END;
         IF level > 0 THEN DevCPT.topScope.link.leaf := FALSE END
      END;
      CheckSym(rparen)
   ELSE EXIT
   END
(*
ELSIF (sym = lparen) & (x.class # Nproc) & (x.typ.form # ProcTyp) &
      ((x.obj = NIL) OR (x.obj.mode # TProc)) THEN
```

```
            DevCPS.Get(sym);
            IF sym = ident THEN
               qualident(obj);
               IF obj.mode = Typ THEN DevCPB.TypTest(x, obj, TRUE)
               ELSE err(52)
               END
            ELSE err(ident)
            END ;
            CheckSym(rparen)
*)
         ELSE EXIT
         END
      END
   END selector;

   PROCEDURE StandProcCall(VAR x: DevCPT.Node);
      VAR y: DevCPT.Node; m: BYTE; n: SHORTINT;
   BEGIN m := SHORT(SHORT(x.obj.adr)); n := 0;
      IF sym = lparen THEN DevCPS.Get(sym);
         IF sym # rparen THEN
            LOOP
               IF n = 0 THEN Expression(x); DevCPB.StPar0(x, m); n := 1
               ELSIF n = 1 THEN Expression(y); DevCPB.StPar1(x, y, m); n := 2
               ELSE Expression(y); DevCPB.StParN(x, y, m, n); INC(n)
               END ;
               IF sym = comma THEN DevCPS.Get(sym)
               ELSIF (lparen <= sym) & (sym <= ident) THEN err(comma)
               ELSE EXIT
               END
            END ;
            CheckSym(rparen)
         ELSE DevCPS.Get(sym)
         END ;
         DevCPB.StFct(x, m, n)
      ELSE err(lparen)
      END ;
      IF (level > 0) & ((m = newfn) OR (m = sysnewfn)) THEN DevCPT.topScope.link.leaf := FALSE END
   END StandProcCall;

   PROCEDURE Element(VAR x: DevCPT.Node);
      VAR y: DevCPT.Node;
   BEGIN Expression(x);
      IF sym = upto THEN
         DevCPS.Get(sym); Expression(y); DevCPB.SetRange(x, y)
      ELSE DevCPB.SetElem(x)
      END
   END Element;

   PROCEDURE Sets(VAR x: DevCPT.Node);
      VAR y: DevCPT.Node;
   BEGIN
      IF sym # rbrace THEN
         Element(x);
         LOOP
            IF sym = comma THEN DevCPS.Get(sym)
            ELSIF (lparen <= sym) & (sym <= ident) THEN err(comma)
            ELSE EXIT
            END ;
            Element(y); DevCPB.Op(plus, x, y)
         END
```

```
      ELSE x := DevCPB.EmptySet()
      END ;
      CheckSym(rbrace)
   END Sets;

   PROCEDURE Factor(VAR x: DevCPT.Node);
      VAR id: DevCPT.Object;
   BEGIN
      IF sym < not THEN err(13);
         REPEAT DevCPS.Get(sym) UNTIL sym >= lparen
      END ;
      IF sym = ident THEN
         qualident(id); x := DevCPB.NewLeaf(id); selector(x);
         IF (x.class = Nproc) & (x.obj.mode = SProc) THEN StandProcCall(x)    (* x may be NIL *)
(*
         ELSIF sym = lparen THEN
            DevCPS.Get(sym); DevCPB.PrepCall(x, fpar);
            ActualParameters(apar, fpar);
            DevCPB.Call(x, apar, fpar);
            CheckSym(rparen);
            IF level > 0 THEN DevCPT.topScope.link.leaf := FALSE END
*)
         END
      ELSIF sym = number THEN
         CASE DevCPS.numtyp OF
           char:
            x := DevCPB.NewIntConst(DevCPS.intval); x.typ := DevCPT.char8typ;
            IF DevCPS.intval > 255 THEN x.typ := DevCPT.char16typ END
         | integer: x := DevCPB.NewIntConst(DevCPS.intval)
         | int64: x := DevCPB.NewLargeIntConst(DevCPS.intval, DevCPS.realval)
         | real: x := DevCPB.NewRealConst(DevCPS.realval, NIL)
         | real32: x := DevCPB.NewRealConst(DevCPS.realval, DevCPT.real32typ)
         | real64: x := DevCPB.NewRealConst(DevCPS.realval, DevCPT.real64typ)
         END ;
         DevCPS.Get(sym)
      ELSIF sym = string THEN
         x := DevCPB.NewString(DevCPS.str, DevCPS.lstr, DevCPS.intval);
         DevCPS.Get(sym)
      ELSIF sym = nil THEN
         x := DevCPB.Nil(); DevCPS.Get(sym)
      ELSIF sym = lparen THEN
         DevCPS.Get(sym); Expression(x); CheckSym(rparen)
      ELSIF sym = lbrak THEN
         DevCPS.Get(sym); err(lparen); Expression(x); CheckSym(rparen)
      ELSIF sym = lbrace THEN DevCPS.Get(sym); Sets(x)
      ELSIF sym = not THEN
         DevCPS.Get(sym); Factor(x); DevCPB.MOp(not, x)
      ELSE err(13); DevCPS.Get(sym); x := NIL
      END ;
      IF x = NIL THEN x := DevCPB.NewIntConst(1); x.typ := DevCPT.undftyp END
   END Factor;

   PROCEDURE Term(VAR x: DevCPT.Node);
      VAR y: DevCPT.Node; mulop: BYTE;
   BEGIN Factor(x);
      WHILE (times <= sym) & (sym <= and) DO
         mulop := sym; DevCPS.Get(sym);
         Factor(y); DevCPB.Op(mulop, x, y)
      END
   END Term;
```

```
PROCEDURE SimpleExpression(VAR x: DevCPT.Node);
   VAR y: DevCPT.Node; addop: BYTE;
BEGIN
   IF sym = minus THEN DevCPS.Get(sym); Term(x); DevCPB.MOp(minus, x)
   ELSIF sym = plus THEN DevCPS.Get(sym); Term(x); DevCPB.MOp(plus, x)
   ELSE Term(x)
   END ;
   WHILE (plus <= sym) & (sym <= or) DO
      addop := sym; DevCPS.Get(sym); Term(y);
      IF x.typ.form = Pointer THEN DevCPB.DeRef(x) END;
      IF (x.typ.comp IN {Array, DynArr}) & (x.typ.BaseTyp.form IN charSet) (* OR (x.typ.sysflag = jstr) *) THEN
         DevCPB.StrDeref(x)
      END;
      IF y.typ.form = Pointer THEN DevCPB.DeRef(y) END;
      IF (y.typ.comp IN {Array, DynArr}) & (y.typ.BaseTyp.form IN charSet) (* OR (y.typ.sysflag = jstr) *) THEN
         DevCPB.StrDeref(y)
      END;
      DevCPB.Op(addop, x, y)
   END
END SimpleExpression;

PROCEDURE Expression(VAR x: DevCPT.Node);
   VAR y, pre, last: DevCPT.Node; obj: DevCPT.Object; relation: BYTE;
BEGIN SimpleExpression(x);
   IF (eql <= sym) & (sym <= geq) THEN
      relation := sym; DevCPS.Get(sym); SimpleExpression(y);
      pre := NIL; last := NIL;
      IF (x.typ.comp IN {Array, DynArr}) & (x.typ.BaseTyp.form IN charSet) THEN
         DevCPB.StrDeref(x)
      END;
      IF (y.typ.comp IN {Array, DynArr}) & (y.typ.BaseTyp.form IN charSet) THEN
         DevCPB.StrDeref(y)
      END;
      DevCPB.CheckBuffering(x, NIL, NIL, pre, last);
      DevCPB.CheckBuffering(y, NIL, NIL, pre, last);
      DevCPB.Op(relation, x, y);
      IF pre # NIL THEN DevCPB.Construct(Ncomp, pre, x); pre.typ := x.typ; x := pre END
   ELSIF sym = in THEN
      DevCPS.Get(sym); SimpleExpression(y); DevCPB.In(x, y)
   ELSIF sym = is THEN
      DevCPS.Get(sym);
      IF sym = ident THEN
         qualident(obj);
         IF obj.mode = Typ THEN DevCPB.TypTest(x, obj, FALSE)
         ELSE err(52)
         END
      ELSE err(ident)
      END
   END
END Expression;

PROCEDURE ProcedureDeclaration(VAR x: DevCPT.Node);
   VAR proc, fwd: DevCPT.Object;
      name: DevCPT.Name;
      mode: BYTE;
      forward: BOOLEAN;
      sys: SHORTINT;

   PROCEDURE GetCode;
      VAR ext: DevCPT.ConstExt; n, c, i: INTEGER; s: POINTER TO ARRAY OF SHORTCHAR;
```

```
      cx: DevCPT.Node;

   PROCEDURE EnsureLen(len: INTEGER);
      VAR j: INTEGER; s2: POINTER TO ARRAY OF SHORTCHAR;
   BEGIN
      IF len > LEN(s) THEN    (* if overflow then increase size of array s *)
         NEW(s2, LEN(s) * 2); FOR j := 0 TO n - 1 DO s2[j] := s[j] END; s := s2
      END
   END EnsureLen;
BEGIN
   n := 0; NEW(s, 64);
   WHILE (sym # semicolon) & (sym # eof) DO
      ConstExpression(cx);
      IF cx.typ.form IN {Int8, Int16, Int32, Char8, Char16} THEN c :=cx.conval.intval; EnsureLen(n + 1);
         IF (0 <= c) & (c <= 255) THEN s[n] := SHORT(CHR(c)); INC(n)
         ELSE err(63)
         END
      ELSIF cx.typ.form = String8 THEN c := cx.conval.intval2 - 1 (*exclude 0X*); EnsureLen(n + c);
         FOR i := 0 TO c - 1 DO s[n + i] := cx.conval.ext[i] END;
         INC(n, c)
      ELSE (* Int64, Real32, Real64, String16, Bool, etc. *) err(63)
      END ;
      IF sym = comma THEN DevCPS.Get(sym); IF sym = semicolon THEN err(13) END
      ELSIF sym # semicolon THEN err(comma)
      END
   END;
   IF n # 0 THEN NEW(ext, n); i := 0;
      WHILE i < n DO ext[i] := s[i]; INC(i) END;
   ELSE ext := NIL
   END;
   proc.conval.ext := ext;
   INCL(proc.conval.setval, hasBody)
END GetCode;

PROCEDURE GetParams;
   VAR name: DevCPT.String;
BEGIN
   proc.mode := mode; proc.typ := DevCPT.notyp;
   proc.sysflag := SHORT(sys);
   proc.conval.setval := {};
   IF sym = lparen THEN
      DevCPS.Get(sym); FormalParameters(proc.link, proc.typ, name);
      IF name # NIL THEN err(0) END
   END;
   CheckForwardTypes; userList := NIL;
   IF fwd # NIL THEN
      DevCPB.CheckParameters(proc.link, fwd.link, TRUE);
      IF ~DevCPT.EqualType(proc.typ, fwd.typ) THEN err(117) END ;
      proc := fwd; DevCPT.topScope := proc.scope;
      IF mode = IProc THEN proc.mode := IProc END
   END
END GetParams;

PROCEDURE Body;
   VAR procdec, statseq: DevCPT.Node; c: INTEGER;
BEGIN
   c := DevCPM.errpos;
   INCL(proc.conval.setval, hasBody);
   CheckSym(semicolon); Block(procdec, statseq);
   DevCPB.Enter(procdec, statseq, proc); x := procdec;
```

```
      x.conval := DevCPT.NewConst(); x.conval.intval := c; x.conval.intval2 := DevCPM.startpos;
      CheckSym(end);
      IF sym = ident THEN
         IF DevCPS.name # proc.name^ THEN err(4) END ;
         DevCPS.Get(sym)
      ELSE err(ident)
      END
   END Body;

   PROCEDURE TProcDecl;
      VAR baseProc, o: DevCPT.Object;
         objTyp, recTyp: DevCPT.Struct;
         objMode, objVis: BYTE;
         objName: DevCPT.Name;
         pnode: DevCPT.Node;
         fwdAttr: SET;
   BEGIN
      DevCPS.Get(sym); mode := TProc;
      IF level > 0 THEN err(73) END;
      Receiver(objMode, objVis, objName, objTyp, recTyp);
      IF sym = ident THEN
         name := DevCPS.name;
         DevCPT.FindField(name, recTyp, fwd);
         DevCPT.FindBaseField(name, recTyp, baseProc);
         IF (baseProc # NIL) & (baseProc.mode # TProc) THEN baseProc := NIL; err(1) END ;
         IF fwd = baseProc THEN fwd := NIL END ;
         IF (fwd # NIL) & (fwd.mnolev # level) THEN fwd := NIL END ;
         IF (fwd # NIL) & (fwd.mode = TProc) & (fwd.conval.setval * {hasBody, absAttr, empAttr} = {}) THEN
            (* there exists a corresponding forward declaration *)
            proc := DevCPT.NewObj(); proc.leaf := TRUE;
            proc.mode := TProc; proc.conval := DevCPT.NewConst();
            CheckMark(proc);
            IF fwd.vis # proc.vis THEN err(118) END;
            fwdAttr := fwd.conval.setval
         ELSE
            IF fwd # NIL THEN err(1); fwd := NIL END ;
            DevCPT.InsertField(name, recTyp, proc);
            proc.mode := TProc; proc.conval := DevCPT.NewConst();
            CheckMark(proc);
            IF recTyp.strobj # NIL THEN    (* preserve declaration order *)
               o := recTyp.strobj.link;
               IF o = NIL THEN recTyp.strobj.link := proc
               ELSE
                  WHILE o.nlink # NIL DO o := o.nlink END;
                  o.nlink := proc
               END
            END
         END;
         INC(level); DevCPT.OpenScope(level, proc);
         DevCPT.Insert(objName, proc.link); proc.link.mode := objMode; proc.link.vis := objVis; proc.link.typ := objTyp;
         ASSERT(DevCPT.topScope # NIL);
         GetParams;    (* may change proc := fwd !!! *)
         ASSERT(DevCPT.topScope # NIL);
         GetAttributes(proc, baseProc, recTyp);
         IF (fwd # NIL) & (fwdAttr / proc.conval.setval * {absAttr, empAttr, extAttr} # {}) THEN err(184) END;
         CheckOverwrite(proc, baseProc, recTyp);
         IF ~forward THEN
            IF empAttr IN proc.conval.setval THEN    (* insert empty procedure *)
               pnode := NIL; DevCPB.Enter(pnode, NIL, proc);
               pnode.conval := DevCPT.NewConst();
```

```
                pnode.conval.intval := DevCPM.errpos;
                pnode.conval.intval2 := DevCPM.errpos;
                x := pnode;
            ELSIF DevCPM.noCode IN DevCPM.options THEN INCL(proc.conval.setval, hasBody)
            ELSIF ~(absAttr IN proc.conval.setval) THEN Body
            END;
            proc.adr := 0
        ELSE
            proc.adr := DevCPM.errpos;
            IF proc.conval.setval * {empAttr, absAttr} # {} THEN err(184) END
        END;
        DEC(level); DevCPT.CloseScope;
    ELSE err(ident)
    END;
END TProcDecl;

BEGIN proc := NIL; forward := FALSE; x := NIL; mode := LProc; sys := 0;
    IF (sym # ident) & (sym # lparen) THEN
        CheckSysFlag(sys, DevCPM.GetProcSysFlag);
        IF sys # 0 THEN
            IF ODD(sys DIV DevCPM.CProcFlag) THEN mode := CProc END
        ELSE
            IF sym = times THEN     (* mode set later in DevCPB.CheckAssign *)
            ELSIF sym = arrow THEN forward := TRUE
            ELSE err(ident)
            END;
            DevCPS.Get(sym)
        END
    END ;
    IF sym = lparen THEN TProcDecl
    ELSIF sym = ident THEN DevCPT.Find(DevCPS.name, fwd);
        name := DevCPS.name;
        IF (fwd # NIL) & ((fwd.mnolev # level) OR (fwd.mode = SProc)) THEN fwd := NIL END ;
        IF (fwd # NIL) & (fwd.mode IN {LProc, XProc}) & ~(hasBody IN fwd.conval.setval) THEN
            (* there exists a corresponding forward declaration *)
            proc := DevCPT.NewObj(); proc.leaf := TRUE;
            proc.mode := mode; proc.conval := DevCPT.NewConst();
            CheckMark(proc);
            IF fwd.vis # proc.vis THEN err(118) END
        ELSE
            IF fwd # NIL THEN err(1); fwd := NIL END ;
            DevCPT.Insert(name, proc);
            proc.mode := mode; proc.conval := DevCPT.NewConst();
            CheckMark(proc);
        END ;
        IF (proc.vis # internal) & (mode = LProc) THEN mode := XProc END ;
        IF (mode # LProc) & (level > 0) THEN err(73) END ;
        INC(level); DevCPT.OpenScope(level, proc);
        proc.link := NIL; GetParams;    (* may change proc := fwd !!! *)
        IF mode = CProc THEN GetCode
        ELSIF DevCPM.noCode IN DevCPM.options THEN INCL(proc.conval.setval, hasBody)
        ELSIF ~forward THEN Body; proc.adr := 0
        ELSE proc.adr := DevCPM.errpos
        END ;
        DEC(level); DevCPT.CloseScope
    ELSE err(ident)
    END
END ProcedureDeclaration;

PROCEDURE CaseLabelList(VAR lab, root: DevCPT.Node; LabelForm: SHORTINT; VAR min, max: INTEGER);
```

```
    VAR x, y: DevCPT.Node; f: SHORTINT; xval, yval: INTEGER;

    PROCEDURE Insert(VAR n: DevCPT.Node);    (* build binary tree of label ranges *)    (* !!! *)
    BEGIN
      IF n = NIL THEN
        IF x.hint # 1 THEN n := x END
      ELSIF yval < n.conval.intval THEN Insert(n.left)
      ELSIF xval > n.conval.intval2 THEN Insert(n.right)
      ELSE err(63)
      END
    END Insert;

  BEGIN lab := NIL;
    LOOP ConstExpression(x); f := x.typ.form;
      IF f IN {Int8..Int32} + charSet THEN  xval := x.conval.intval
      ELSE err(61); xval := 1
      END ;
      IF (f IN {Int8..Int32}) # (LabelForm IN {Int8..Int32}) THEN err(60) END;
      IF sym = upto THEN
        DevCPS.Get(sym); ConstExpression(y); yval := y.conval.intval;
        IF (y.typ.form IN {Int8..Int32}) # (LabelForm IN {Int8..Int32}) THEN err(60) END;
        IF yval < xval THEN err(63); yval := xval END
      ELSE yval := xval
      END ;
      x.conval.intval2 := yval;
      IF xval < min THEN min := xval END;
      IF yval > max THEN max := yval END;
      IF lab = NIL THEN lab := x; Insert(root)
      ELSIF yval < lab.conval.intval - 1 THEN x.link := lab; lab := x; Insert(root)
      ELSIF yval = lab.conval.intval - 1 THEN x.hint := 1; Insert(root); lab.conval.intval := xval
      ELSIF xval = lab.conval.intval2 + 1 THEN x.hint := 1; Insert(root); lab.conval.intval2 := yval
      ELSE
        y := lab;
        WHILE (y.link # NIL) & (xval > y.link.conval.intval2 + 1) DO y := y.link END;
        IF y.link = NIL THEN y.link := x; Insert(root)
        ELSIF yval < y.link.conval.intval - 1 THEN x.link := y.link; y.link := x; Insert(root)
        ELSIF yval = y.link.conval.intval - 1 THEN x.hint := 1; Insert(root); y.link.conval.intval := xval
        ELSIF xval = y.link.conval.intval2 + 1 THEN x.hint := 1; Insert(root); y.link.conval.intval2 := yval
        END
      END;
      IF sym = comma THEN DevCPS.Get(sym)
      ELSIF (sym = number) OR (sym = ident) THEN err(comma)
      ELSE EXIT
      END
    END
  END CaseLabelList;

  PROCEDURE StatSeq(VAR stat: DevCPT.Node);
    VAR fpar, id, t: DevCPT.Object; idtyp: DevCPT.Struct; e: BOOLEAN;
        s, x, y, z, apar, last, lastif, pre, lastp: DevCPT.Node; pos, p: INTEGER;

    PROCEDURE CasePart(VAR x: DevCPT.Node);
      VAR low, high: INTEGER; e: BOOLEAN; cases, lab, y, lastcase, root: DevCPT.Node;
    BEGIN
      Expression(x);
      IF (x.class = Ntype) OR (x.class = Nproc) THEN err(126)
      ELSIF x.typ.form = Int64 THEN err(260)
      ELSIF ~(x.typ.form IN {Int8..Int32} + charSet) THEN err(125)
      END ;
      CheckSym(of); cases := NIL; lastcase := NIL; root := NIL;
```

```
      low := MAX(INTEGER); high := MIN(INTEGER);
      LOOP
        IF sym < bar THEN
          CaseLabelList(lab, root, x.typ.form, low, high);
          CheckSym(colon); StatSeq(y);
          DevCPB.Construct(Ncasedo, lab, y); DevCPB.Link(cases, lastcase, lab)
        END ;
        IF sym = bar THEN DevCPS.Get(sym) ELSE EXIT END
      END;
      e := sym = else;
      IF e THEN DevCPS.Get(sym); StatSeq(y) ELSE y := NIL END ;
      DevCPB.Construct(Ncaselse, cases, y); DevCPB.Construct(Ncase, x, cases);
      cases.conval := DevCPT.NewConst();
      cases.conval.intval := low; cases.conval.intval2 := high;
      IF e THEN cases.conval.setval := {1} ELSE cases.conval.setval := {} END;
      DevCPB.OptimizeCase(root); cases.link := root    (* !!! *)
    END CasePart;

    PROCEDURE SetPos(x: DevCPT.Node);
    BEGIN
      x.conval := DevCPT.NewConst(); x.conval.intval := pos
    END SetPos;

    PROCEDURE CheckBool(VAR x: DevCPT.Node);
    BEGIN
      IF (x.class = Ntype) OR (x.class = Nproc) THEN err(126); x := DevCPB.NewBoolConst(FALSE)
      ELSIF x.typ.form # Bool THEN err(120); x := DevCPB.NewBoolConst(FALSE)
      END
    END CheckBool;

  BEGIN stat := NIL; last := NIL;
    LOOP x := NIL;
      IF sym < ident THEN err(14);
        REPEAT DevCPS.Get(sym) UNTIL sym >= ident
      END ;
      pos := DevCPM.startpos;
      IF sym = ident THEN
        qualident(id); x := DevCPB.NewLeaf(id); selector(x);
        IF sym = becomes THEN
          DevCPS.Get(sym); Expression(y);
          IF (y.typ.form = Pointer) & (x.typ.form = Comp) THEN DevCPB.DeRef(y) END;
          pre := NIL; lastp := NIL;
          DevCPB.CheckBuffering(y, x, NIL, pre, lastp);
          DevCPB.Assign(x, y);
          IF pre # NIL THEN SetPos(x); DevCPB.Construct(Ncomp, pre, x); x := pre END;
        ELSIF sym = eql THEN
          err(becomes); DevCPS.Get(sym); Expression(y); DevCPB.Assign(x, y)
        ELSIF (x.class = Nproc) & (x.obj.mode = SProc) THEN
          StandProcCall(x);
          IF (x # NIL) & (x.typ # DevCPT.notyp) THEN err(55) END;
          IF (x # NIL) & (x.class = Nifelse) THEN    (* error pos for ASSERT *)
            SetPos(x.left); SetPos(x.left.right)
          END
        ELSIF x.class = Ncall THEN err(55)
        ELSE
          pre := NIL; lastp := NIL;
          DevCPB.PrepCall(x, fpar);
          IF (x.obj # NIL) & (x.obj.mode = TProc) THEN DevCPB.CheckBuffering(x.left, NIL, x.obj.link, pre, lastp) END;
          IF sym = lparen THEN
            DevCPS.Get(sym); ActualParameters(apar, fpar, pre, lastp); CheckSym(rparen)
```

```
        ELSE apar := NIL;
          IF fpar # NIL THEN err(65) END
        END ;
        DevCPB.Call(x, apar, fpar);
        IF x.typ # DevCPT.notyp THEN err(55) END;
        IF pre # NIL THEN SetPos(x); DevCPB.Construct(Ncomp, pre, x); x := pre END;
        IF level > 0 THEN DevCPT.topScope.link.leaf := FALSE END
      END
    ELSIF sym = if THEN
      DevCPS.Get(sym); pos := DevCPM.startpos; Expression(x); CheckBool(x); CheckSym(then); StatSeq(y);
      DevCPB.Construct(Nif, x, y); SetPos(x); lastif := x;
      WHILE sym = elsif DO
        DevCPS.Get(sym); pos := DevCPM.startpos; Expression(y); CheckBool(y); CheckSym(then); StatSeq(z);
        DevCPB.Construct(Nif, y, z); SetPos(y); DevCPB.Link(x, lastif, y)
      END ;
      pos := DevCPM.startpos;
      IF sym = else THEN DevCPS.Get(sym); StatSeq(y) ELSE y := NIL END ;
      DevCPB.Construct(Nifelse, x, y); CheckSym(end); DevCPB.OptIf(x);
    ELSIF sym = case THEN
      DevCPS.Get(sym); pos := DevCPM.startpos; CasePart(x); CheckSym(end)
    ELSIF sym = while THEN
      DevCPS.Get(sym); pos := DevCPM.startpos; Expression(x); CheckBool(x); CheckSym(do); StatSeq(y);
      DevCPB.Construct(Nwhile, x, y); CheckSym(end)
    ELSIF sym = repeat THEN
      DevCPS.Get(sym); StatSeq(x);
      IF sym = until THEN DevCPS.Get(sym); pos := DevCPM.startpos; Expression(y); CheckBool(y)
      ELSE err(43)
      END ;
      DevCPB.Construct(Nrepeat, x, y)
    ELSIF sym = for THEN
      DevCPS.Get(sym); pos := DevCPM.startpos;
      IF sym = ident THEN qualident(id);
        IF ~(id.typ.form IN intSet) THEN err(68) END ;
        CheckSym(becomes); Expression(y);
        x := DevCPB.NewLeaf(id); DevCPB.Assign(x, y); SetPos(x);
        CheckSym(to); pos := DevCPM.startpos; Expression(y);
        IF y.class # Nconst THEN
          DevCPB.GetTempVar("@for", x.left.typ, t);
          z := DevCPB.NewLeaf(t); DevCPB.Assign(z, y); SetPos(z); DevCPB.Link(stat, last, z);
          y := DevCPB.NewLeaf(t)
        ELSE
          DevCPB.CheckAssign(x.left.typ, y)
        END ;
        DevCPB.Link(stat, last, x);
        p := DevCPM.startpos;
        IF sym = by THEN DevCPS.Get(sym); ConstExpression(z) ELSE z := DevCPB.NewIntConst(1) END ;
        x := DevCPB.NewLeaf(id);
        IF z.conval.intval > 0 THEN DevCPB.Op(leq, x, y)
        ELSIF z.conval.intval < 0 THEN DevCPB.Op(geq, x, y)
        ELSE err(63); DevCPB.Op(geq, x, y)
        END ;
        CheckSym(do); StatSeq(s);
        y := DevCPB.NewLeaf(id); DevCPB.StPar1(y, z, incfn); pos := DevCPM.startpos; SetPos(y);
        IF s = NIL THEN s := y
        ELSE z := s;
          WHILE z.link # NIL DO z := z.link END ;
          z.link := y
        END ;
        CheckSym(end); DevCPB.Construct(Nwhile, x, s); pos := p
      ELSE err(ident)
```

```
            END
        ELSIF sym = loop THEN
            DevCPS.Get(sym); INC(LoopLevel); StatSeq(x); DEC(LoopLevel);
            DevCPB.Construct(Nloop, x, NIL); CheckSym(end)
        ELSIF sym = with THEN
            DevCPS.Get(sym); idtyp := NIL; x := NIL;
            LOOP
                IF sym < bar THEN
                    pos := DevCPM.startpos;
                    IF sym = ident THEN
                        qualident(id); y := DevCPB.NewLeaf(id);
                        IF (id # NIL) & (id.typ.form = Pointer) & ((id.mode = VarPar) OR ~id.leaf) THEN
                            err(-302)    (* warning 302 *)
                        END ;
                        CheckSym(colon);
                        IF sym = ident THEN qualident(t);
                            IF t.mode = Typ THEN
                                IF id # NIL THEN
                                    idtyp := id.typ; DevCPB.TypTest(y, t, FALSE); id.typ := t.typ;
                                    IF id.ptyp = NIL THEN id.ptyp := idtyp END
                                ELSE err(130)
                                END
                            ELSE err(52)
                            END
                        ELSE err(ident)
                        END
                    ELSE err(ident)
                    END ;
                    CheckSym(do); StatSeq(s); DevCPB.Construct(Nif, y, s); SetPos(y);
                    IF idtyp # NIL THEN
                        IF id.ptyp = idtyp THEN id.ptyp := NIL END;
                        id.typ := idtyp; idtyp := NIL
                    END ;
                    IF x = NIL THEN x := y; lastif := x ELSE DevCPB.Link(x, lastif, y) END
                END;
                IF sym = bar THEN DevCPS.Get(sym) ELSE EXIT END
            END;
            e := sym = else; pos := DevCPM.startpos;
            IF e THEN DevCPS.Get(sym); StatSeq(s) ELSE s := NIL END ;
            DevCPB.Construct(Nwith, x, s); CheckSym(end);
            IF e THEN x.subcl := 1 END
        ELSIF sym = exit THEN
            DevCPS.Get(sym);
            IF LoopLevel = 0 THEN err(46) END ;
            DevCPB.Construct(Nexit, x, NIL)
        ELSIF sym = return THEN DevCPS.Get(sym);
            IF sym < semicolon THEN Expression(x) END ;
            IF level > 0 THEN DevCPB.Return(x, DevCPT.topScope.link)
            ELSE (* not standard Oberon *) DevCPB.Return(x, NIL)
            END;
            hasReturn := TRUE
        END ;
        IF x # NIL THEN SetPos(x); DevCPB.Link(stat, last, x) END ;
        IF sym = semicolon THEN DevCPS.Get(sym)
        ELSIF (sym <= ident) OR (if <= sym) & (sym <= return) THEN err(semicolon)
        ELSE EXIT
        END
    END
END StatSeq;
```

```
PROCEDURE Block(VAR procdec, statseq: DevCPT.Node);
   VAR typ: DevCPT.Struct;
      obj, first, last, o: DevCPT.Object;
      x, lastdec: DevCPT.Node;
      i: SHORTINT;
      rname: DevCPT.Name;
      name: DevCPT.String;
      rec: Elem;

BEGIN
   IF ((sym < begin) OR (sym > var)) & (sym # procedure) & (sym # end) & (sym # close) THEN err(36) END;
   first := NIL; last := NIL; userList := NIL; recList := NIL;
   LOOP
      IF sym = const THEN
         DevCPS.Get(sym);
         WHILE sym = ident DO
            DevCPT.Insert(DevCPS.name, obj);
            obj.mode := Con; CheckMark(obj);
            obj.typ := DevCPT.int8typ; obj.mode := Var;    (* Var to avoid recursive definition *)
            IF sym = eql THEN
               DevCPS.Get(sym); ConstExpression(x)
            ELSIF sym = becomes THEN
               err(eql); DevCPS.Get(sym); ConstExpression(x)
            ELSE err(eql); x := DevCPB.NewIntConst(1)
            END ;
            obj.mode := Con; obj.typ := x.typ; obj.conval := x.conval; (* ConstDesc is not copied *)
            CheckSym(semicolon)
         END
      END ;
      IF sym = type THEN
         DevCPS.Get(sym);
         WHILE sym = ident DO
            DevCPT.Insert(DevCPS.name, obj); obj.mode := Typ; obj.typ := DevCPT.undftyp;
            CheckMark(obj); obj.mode := -1;
            IF sym # eql THEN err(eql) END;
            IF (sym = eql) OR (sym = becomes) OR (sym = colon) THEN
               DevCPS.Get(sym); Type(obj.typ, name); SetType(NIL, obj, obj.typ, name);
            END;
            obj.mode := Typ;
            IF obj.typ.form IN {Byte..Set, Char16, Int64} THEN    (* make alias structure *)
               typ := DevCPT.NewStr(obj.typ.form, Basic); i := typ.ref;
               typ^ := obj.typ^; typ.ref := i; typ.strobj := NIL; typ.mno := 0; typ.txtpos := DevCPM.errpos;
               typ.BaseTyp := obj.typ; obj.typ := typ;
            END;
            IF obj.typ.strobj = NIL THEN obj.typ.strobj := obj END ;
            IF obj.typ.form = Pointer THEN    (* !!! *)
               typ := obj.typ.BaseTyp;
               IF (typ # NIL) & (typ.comp = Record) & (typ.strobj = NIL) THEN
                  (* pointer to unnamed record: name record as "pointerName^" *)
                  rname := obj.name$; i := 0;
                  WHILE rname[i] # 0X DO INC(i) END;
                  rname[i] := "^"; rname[i+1] := 0X;
                  DevCPT.Insert(rname, o); o.mode := Typ; o.typ := typ; typ.strobj := o
               END
            END;
            IF obj.vis # internal THEN
               typ := obj.typ;
               IF typ.form = Pointer THEN typ := typ.BaseTyp END;
               IF typ.comp = Record THEN typ.exp := TRUE END
            END;
```

```
            CheckSym(semicolon)
          END
        END ;
        IF sym = var THEN
          DevCPS.Get(sym);
          WHILE sym = ident DO
            LOOP
              IF sym = ident THEN
                DevCPT.Insert(DevCPS.name, obj);
                obj.mode := Var; obj.link := NIL; obj.leaf := obj.vis = internal; obj.typ := DevCPT.undftyp;
                CheckMark(obj);
                IF first = NIL THEN first := obj END ;
                IF last = NIL THEN DevCPT.topScope.scope := obj ELSE last.link := obj END ;
                last := obj
              ELSE err(ident)
              END ;
              IF sym = comma THEN DevCPS.Get(sym)
              ELSIF sym = ident THEN err(comma)
              ELSE EXIT
              END
            END ;
            CheckSym(colon); Type(typ, name);
            CheckAlloc(typ, FALSE, DevCPM.errpos);
            WHILE first # NIL DO SetType(NIL, first, typ, name); first := first.link END ;
            CheckSym(semicolon)
          END
        END ;
        IF (sym < const) OR (sym > var) THEN EXIT END ;
      END ;
      CheckForwardTypes;
      userList := NIL; rec := recList; recList := NIL;
      DevCPT.topScope.adr := DevCPM.errpos;
      procdec := NIL; lastdec := NIL;
      IF (sym # procedure) & (sym # begin) & (sym # end) & (sym # close) THEN err(37) END;
      WHILE sym = procedure DO
        DevCPS.Get(sym); ProcedureDeclaration(x);
        IF x # NIL THEN
          IF lastdec = NIL THEN procdec := x ELSE lastdec.link := x END ;
          lastdec := x
        END ;
        CheckSym(semicolon)
      END ;
      IF DevCPM.noerr & ~(DevCPM.oberon IN DevCPM.options) THEN CheckRecords(rec) END;
      hasReturn := FALSE;
      IF (sym # begin) & (sym # end) & (sym # close) THEN err(38) END;
      IF sym = begin THEN DevCPS.Get(sym); StatSeq(statseq)
      ELSE statseq := NIL
      END ;
      IF (DevCPT.topScope.link # NIL) & (DevCPT.topScope.link.typ # DevCPT.notyp)
        & ~hasReturn & (DevCPT.topScope.link.sysflag = 0) THEN err(133) END;
      IF (level = 0) & (TDinit # NIL) THEN
        lastTDinit.link := statseq; statseq := TDinit
      END
    END Block;

    PROCEDURE Module*(VAR prog: DevCPT.Node);
      VAR impName, aliasName: DevCPT.Name;
          procdec, statseq: DevCPT.Node;
          c: INTEGER; done: BOOLEAN;
    BEGIN
```

```
		DevCPS.Init; LoopLevel := 0; level := 0; DevCPS.Get(sym);
		IF sym = module THEN DevCPS.Get(sym) ELSE err(16) END ;
		IF sym = ident THEN
			DevCPT.Open(DevCPS.name); DevCPS.Get(sym);
			DevCPT.libName := "";
			IF sym = lbrak THEN
				INCL(DevCPM.options, DevCPM.interface); DevCPS.Get(sym);
				IF sym = eql THEN DevCPS.Get(sym)
				ELSE INCL(DevCPM.options, DevCPM.noCode)
				END;
				IF sym = string THEN DevCPT.libName := DevCPS.str$; DevCPS.Get(sym)
				ELSE err(string)
				END;
				CheckSym(rbrak)
			END;
			CheckSym(semicolon);
			IF sym = import THEN DevCPS.Get(sym);
				LOOP
					IF sym = ident THEN
						aliasName := DevCPS.name$; impName := aliasName$; DevCPS.Get(sym);
						IF sym = becomes THEN DevCPS.Get(sym);
							IF sym = ident THEN impName := DevCPS.name$; DevCPS.Get(sym) ELSE err(ident) END
						END ;
						DevCPT.Import(aliasName, impName, done)
					ELSE err(ident)
					END ;
					IF sym = comma THEN DevCPS.Get(sym)
					ELSIF sym = ident THEN err(comma)
					ELSE EXIT
					END
				END ;
				CheckSym(semicolon)
			END ;
			IF DevCPM.noerr THEN TDinit := NIL; lastTDinit := NIL; c := DevCPM.errpos;
				Block(procdec, statseq); DevCPB.Enter(procdec, statseq, NIL); prog := procdec;
				prog.conval := DevCPT.NewConst(); prog.conval.intval := c; prog.conval.intval2 := DevCPM.startpos;
				IF sym = close THEN DevCPS.Get(sym); StatSeq(prog.link) END;
				prog.conval.realval := DevCPM.startpos;
				CheckSym(end);
				IF sym = ident THEN
					IF DevCPS.name # DevCPT.SelfName THEN err(4) END ;
					DevCPS.Get(sym)
				ELSE err(ident)
				END;
				IF sym # period THEN err(period) END
			END
		ELSE err(ident)
		END ;
		TDinit := NIL; lastTDinit := NIL;
		DevCPS.str := NIL
	END Module;

END DevCPP.
```