```
   manifest
   §m
       PreviousLevel[L] = L↓0
       LevelType[L] = L↓1
       Generation[L] = L↓2
       NameList[L] = L↓3

       UpdatePreviousLevel[L, P] is L↓0 := P
       UpdateLevelType[L, T] is L↓1 := T
       UpdateGeneration[L, G] is L↓2 := G
       UpdateNameList[L, N] is L↓3 := N

       PreviousNameBlock[N] = N↓0
       NameType[N] = N↓1
    || Generation[N] = N↓2
       NextName[N] = N↓3
       NameVal[N] = N↓4
       NameNo[N] = N↓5
       NameField[N] = N↓6

       UpdatePreviousNameBlock[N, P] is N↓0 := P
       UpdateNameType[N, T] is N↓1 := T
    || UpdateGeneration[N, G] is N↓2 := G
       UpdateNextName[N, V] is N↓3 := V
       UpdateNameVal[N, V] is N↓4 := V
       UpdateNameNo[N, V] is N↓5 := V
    || UpdateNameField[N, F] is N↓6 := F

       NameBlock[N] = NameVec↓(ValPart[N])
       UpdateNameBlock[N, NB] is NameVec↓(ValPart[N]) := NB

       IsCType[C, T] = (C ∧ TYPEMASK) = T

       UpdateCh[C] is
       §U
           Ch := C
           ChType := TypePart[C]
       $U

       BackUp[C] is
       §BU
           PutBack[In, Ch]
           UpdateCh[C]
       $BU

       Insert[C] is
       §C
           Send[Ch]
           UpdateCh[C]
       $C


       MayPrecede[x, y] = Generation[x] < Generation[y]
$m

   let SetLevel[L] be
   §SL
       let A = CommonAncestor[PresentLevel, L]
       until PresentLevel = A do UpOneLevel[]
       DowntoLevel[L]
```

```
    $SL

    and UpOneLevel[] be
65  §UOL
        let p = NameList[PresentLevel]
        until p = NILNAME do
        §   UpdateNameBlock[NameField[p], PreviousNameBlock[p]]
            p := NextName[p]
70      §
        PresentLevel := PreviousLevel[PresentLevel]
    $UOL

    and DowntoLevel[L] be
75      unless PresentLevel = L do
        §DL
            DowntoLevel[PreviousLevel[L]]
          § let p = NameList[L]
            until p = NILNAME do
80          §   UpdatePreviousNameBlock[p, NameBlock[NameField[p]]]
                UpdateNameBlock[NameField[p], p]
                p := NextName[p]
            §
            PresentLevel := L
85      $DL

    and CommonAncestor[x, y] = (x = y) → x,
        MayPrecede[x, y] → CommonAncestor[x, PreviousLevel[y]],
                           CommonAncestor[PreviousLevel[x], y]
90

    let Read[] be
    §R
        UpdateCh[Next[In]]
95      switchon ChType into
        §S
            case NEWLINE:
                    LineNo := ValPart[Ch]
                    endcase
100
            case GET:
                    GetNo := ValPart[Ch]
                    endcase

105         case NEWLEVEL:
                    SetLevel[StructureVec + ValPart[Ch]]
                    endcase

            case TYPEMASK:
110                 Error[5, HARD, DUMMY, Finish]

            case DUMMY:
                    loop

115         default:return
        $S
        SpecSend[Ch]
    $R repeat

120 and Scan[] be
    §S
        Send[Ch]
        Read[]
    $S
125
```

****