```
   let Pass3[] be
   §P3
       PassNo := 3
       Send := FailSend
       SpecSend := NullProgram
       LineNo := 1
       GetNo := 0
       ReportNo := 0
       LocalGenNo := 0
       HN := 0
       SWN := 0
       TableMax := 0
       NumTable := SetUpNumTable[]
       Stack := NewVec[STACKSIZE]
       Stack↓SIZE := STACKSIZE
       SetUp3b[]
       Peep[1]

       Write[ENDPROG]
       Command[]

       CloseDown3b[]
       ReturnVec[Stack, STACKSIZE]
       ReturnVec[NumTable, NumTable↓SIZE]
       ReturnVec[StructureVec, StructureSize]
       ReturnVec[NameVec, NameSize + 26]
       Peep[3]
   §P3


   and LeftName[C] be
   §LN
       let F = FindName[C]
       switchon NameType[F] into
       §s
           case PARAM:
           case SIMPLE:
                   Code[LPsm, NameVal[F]]
                   return

           case GLOBAL:
                   Code[LGsm, NameVal[F]]
                   return

           case STATIC:
           case TABLE:
           case LABEL:
                   LoadStatic[Nsm, NameNo[F]]
                   return

           case MANIFEST:
           case MANFN:
           case MANRT:
                   Error[204, HARD, C, NullProgram]

           case UNDEFINED:
                   CodeNsm[0]
                   return

           default:CompilerError[3101]
```

```
        §s
    §LN

65
    and RightName[C] be
    §RN
        let F = FindName[C]
        switchon NameType[F] into
70      §s
            case PARAM:
            case SIMPLE:
                    Code[RPsm, NameVal[F]]
                    return
75
            case GLOBAL:
                    Code[RGsm, NameVal[F]]
                    return

80          case STATIC:
            case TABLE:
            case LABEL:
                    LoadStatic[RAsm, NameNo[F]]
                    return
85
            case MANIFEST:
                    CodeNsm[NameVal[F]]
                    return

90          case MANFN:
            case MANRT:
                    if UNDEFINED ≠ NameVal[F] ≠ UNSET do
                    §   ClearStack[]
                        Addto[OutputVec, MFAPP]
95                      ChangeOutput[]
                    §
                    LoadManFun[NameNo[F], NameVal[F]]
                    return

100         case UNDEFINED:
                    CodeNsm[0]
                    return

            default:CompilerError[3102]
105     §s
    §RN


    and StoreName[C] be
110 §SN
        let F = FindName[C]
        switchon NameType[F] into
        §s
            case PARAM:
115         case SIMPLE:
                    Code[SPsm, NameVal[F]]
                    return

            case GLOBAL:
120                 Code[SGsm, NameVal[F]]
                    return

            case STATIC:
            case TABLE:
125         case LABEL:
                    LoadStatic[SAsm, NameNo[F]]
```

```
                    return

            case MANIFEST:
130         case MANFN:
            case MANRT:
                    Error[204, HARD, C, NullProgram]

            case UNDEFINED:
135                 Code[SPsm, 0]
                    return

            default:CompilerError[3103]
        $s
140 $SN


    and LeftVersion[C, CType] be
    §LV
145     switchon CType into
        §s
            case VECAP:
            case VECOP:
                    InstP[ADDsm]
150
            case RV:return

            case NAME:
                    LeftName[C]
155                 return

            default:Error[181, HARD, C, NullProgram]
        $s
    $LV
160

    and RightVersion[C, CType] be
    §RV
        switchon CType into
165     §s
            case NAME:
                    RightName[C]
                    return

170         case NUMBER:
                    CodeNsm[NumberVal[C]]
                    return

            case CHARACTER:
175                 CodeNsm[ValPart[C]]
                    return

            case STRING:
                    LoadString[ValPart[C]]
180                 return

            case TRUE:
                    CodeNsm[true]
                    return
185
            case FALSE:
                    CodeNsm[false]
                    return

190         case UPLUS:
            case DUMMY:
```

```
                return

        case NOT:
195             InstM[NOTsm]
                return

        case UMINUS:
                InstM[NEGsm]
200             return

        case PLUS:
                InstP[ADDsm]
                return
205
        case MINUS:
                InstP[SUBsm]
                return

210     case MULT:
                InstD[MULTsm]
                return

        case DIV:
215             InstD[DIVsm]
                return

        case REM:
                InstD[REMsm]
220             return

        case LSHIFT:
                InstD[LSHsm]
                return
225
        case RSHIFT:
                InstD[RSHsm]
                return

230     case LOGAND:
                InstD[ANDsm]
                return

        case LOGOR:
235             InstD[ORsm]
                return

        case EQV:
                InstD[EQVsm]
240             return

        case NEQV:
                InstD[NEQVsm]
                return
245
        case RV:Inst[CONTSsm]
                return

        case VECAP:
250     case VECOP:
                Inst[VECAPsm]
                return

        case EQ:
255     case NE:
        case LT:
```

```
            case LE:
            case GT:
            case GE:InstRel[CType]
260                 return

            default:CompilerError[3104]

            case LV:
265         case ASS:
            case COND:

        $s
    $RV
270

    and StoreVersion[C, CType] be
    §SV
        switchon CType into
275     §s
            case VECAP:
            case VECOP:
                    Inst[VECSTsm]
                    return
280
            case RV:Inst[STOREsm]
                    return

            case NAME:
285                 StoreName[C]
                    return

            default:Error[181, HARD, C, NullProgram]
        §s
290 §SV


    and CodeNsm[n] be
    §CN
295     test StackPtr ≥ Stack↓SIZE
          ifso  §   NormalOutput[2, Nsm, Stack↓(StackBase + 1)]
                    Copy[Stack + StackBase + 2, Stack + StackBase + 1,
                            STACKSIZE - 1 - StackBase]
                    Stack↓STACKSIZE := n
300               §
          ifnot §   StackPtr := StackPtr + 1
                    Stack↓StackPtr := n
                §
        SSP := SSP + 1
305 §CN


    and InstD[Type] be
    §ID
310     test StackSize[] ≥ 2
          ifso  StackOp[Type]
          ifnot Inst[Type]
    §ID

315
    and InstM[Type] be
    §IM
        test StackSize[] ≥ 1
          ifso  §   let v = Stack↓StackPtr
320                 Stack↓StackPtr := Type = NOTsm → v, -v
                §
```

```
          ifnot Inst[Type]
    §IM

325
    and InstP[Type] be
    §IP
        switchon StackSize[] into
        §s
330         case 0: Inst[Type]
                    return

            case 1:
                  § let n = (Type = SUBsm → -Stack↓StackPtr, Stack↓StackPtr)
335                 StackPtr := StackBase
                    NormalOutput[2, INCsm, n]
                    SSP := SSP - 1
                    return
                  §
340
            default:StackOp[Type]
                    return
        §s
    §IP
345

    and InstRel[Type] be
    §IR
        unless StackSize[] ≥ 2 do CompilerError[3105]
350     StackOp[Type]
    §IR


    and StackOp[Type] be
355 §SO
        let p = StackPtr - 1
        Stack↓p := ConstOp[Type, Stack↓p, Stack↓StackPtr]
        StackPtr := p
        SSP := SSP - 1
360 §SO


    and NumberVal[C] = valof
    §   let n = ValPart[C]
365     resultis 0 ≤ n ≤ 500 → n, NumTable↓(n - 500)
    §


    and SetUpNumTable[] = valof
370 §SNT
        let Table = NewVec[NumberSize]
        let S = InfromFile[WorkFile['Numbers']]
        let n = Next[S]
        Table↓SIZE := NumberSize
375     TransferIn[S, Table + 1, n]
        Close[S]
        resultis Table
    §SNT

380
    and StackOutput[p] be
    §SO
        switchon CellType[p] into
        §s
385         case Nsm:
                    CodeNsm[CellNo[p]]
```

```
                endcase

        case ADDsm:
390     case SUBsm:
                InstP[CellType[p]]
                endcase

        case NOTsm:
395     case NEGsm:
                InstM[CellType[p]]
                endcase

        case MULTsm:
400     case DIVsm:
        case REMsm:
        case LSHsm:
        case RSHsm:
        case ANDsm:
405     case ORsm:
        case EQVsm:
        case NEQVsm:
                InstD[CellType[p]]
                endcase
410
        case INCsm:
                if StackSize[] > 0 do
                §   Stack↓StackPtr := Stack↓StackPtr + CellNo[p]
                    endcase
415             §

        default:ClearStack[]
                CellOutput[p]
                endcase
420 $s
$SO


and ClearStack[] be
425 if StackPtr > StackBase do
    §
        let p = StackBase
        StackBase := StackPtr
        for i = p + 1 to StackPtr do
430         test Stack↓i = -1
              ifso  NormalOutput[1, TRUEsm]
              ifnot NormalOutput[2, Nsm, Stack↓i]
        ClearStack[]
        StackPtr, StackBase := p, p
435 §


and FailSend[x] be
    CompilerError[3106]
440

****
```