```
        { ************************************************** }
        { ***** Unidad de Analisis Lexicografico de QENQO **** }
        { ************************************************** }
        { * Autor: JOSE LUIS SEGOVIA JUAREZ * }
        { * Lima - Peru *. 1991 * }

UNIT lexanuni;
INTERFACE
USES crt, graph;

        { ** Definiciones y Declaraciones de todo el Programa *}
        { ** Definicion de Constantes ** }
CONST
   MAXTS     = 350;       {* Numero de registros en Tabla Simbolos *}
   MAXPALRES = 73;        {* Numero de Palabras reservadas *}
   maxerr    = 90;        {* Numero de errores *}
   toperror  = 10;

     { * Constantes para signos * }
   CR        = #13;
   PAR_I     = 40;   PAR_D  =  41;    CORCH_I   = 91; CORCH_D = 93;
   COMA      = 44;   PUNTO_COMA = 59; APOSTROFO = 39; MUL     = 42;
   DIVIS     = 47;   MENOS      = 45; IGUAL     = 61; MENOR   = 60;
   MAYOR     = 62;   PUNTO      = 46; MAS       = 43; DISTINTO= 1100;
   PUNTERO   = 94;

     { * Constantes para signos compuestos e identificadores * }
   FIN       = 0;
   IDENT     = 1000;
   NUM_I     = 1010;
   NUM_R     = 1020;
   LITERAL   = 1025;
   ASIG      = 1030;
   DOS_PUNTOS_H = 2010;  MAYOR_IGUAL = 1110; MENOR_IGUAL = 1120;
   DOS_PUNTOS_V = 2000;  NUEVA_LINEA = 1130;

     { ** Constantes del Codigo Intermedio ** }
   JMP         = 320;
   JE          = 321;
   JNE         = 322;
   JL          = 323;
   JG          = 324;
   JLE         = 325;
   JGE         = 326;
   CALL        = 327;
   RET         = 328;
   JNZ         = 329;
   PUSH        = 330;
   POP         = 331;
   MOV         = 332;
   WRT         = 333;
   WRTLN       = 334;
   ASGI        = 335;
   ABAR        = 336;
   CEAR        = 337;
   WRAR        = 338;
   ASAT        = 339;
```

```
VE          = 341;
VNE         = 342;
VL          = 343;
VG          = 344;
VLE         = 345;
VGE         = 346;
VNZ         = 347;
INTP        = 348;
INTX        = 349;

  {** Constantes de la Tabla de Simbolos  **}

tSISTEMA     =1300;
tCONS        =1310;
tVAR         =1320;
tNIVEL       =1330;
tFLUJO       =1340;
tAUX         =1380;
tTABLA       =1390;
tCONTROL     =1400;
tDT          =1410;
tDESDE_T     =1420;
tHASTA_T     =1430;
tLONGITUD    =1440;
tRANGOS      =1450;
tOBJETIVO    =1460;
tSUBSIS      =1470;
tFUNCION     =1480;
tINICIO      =1490;
tFIN         =1500;
tSI          =1510;
tENTONCES    =1520;
tSINO        =1530;
tMIENTRAS    =1540;
tHACER       =1550;
tDESDE       =1560;
tHASTA       =1570;
tRHASTA      =1580;
tNO          =1590;
tOR          =1600;
tY           =1610;
tESCRIBE     =1620;   tINTERPOLA = 1680;
tESCRIBELN   =1630;   tINTERPOLX = 1690;
tGRAFICA     =1650;
tPON_GRAFICA =1660;
tTIEMPO      =1670;

   { ** Constantes para las variables y atributos ** }

ID_NIVEL     = tNIVEL;
ID_TIPO      = 302;
ID_SISTEMA   = 303;
ID_CTE       = 304;
ID_VAR       = 305;
ID_SUBSIS    = 306;
ID_FUNC      = 307;
ID_CONS      = 308;
ID_AUX       = tAUX;
```

```
  ID_TABLA      = tTABLA;
  ID_FLUJO      = tFLUJO;
  ID_NUM        = 312;
  ID_CONTROL    = 313;
  val_tabla     = 314;

  maxcode = 350;          { * Num de Tabla de Codigo Intermedio  * }
  maxinterp = 10;
  maxcte  = 1;
TYPE
  tipotempo  = STRING[80];   { ** Cadena de lectura de codigo fuente ** }
  tiponombre = STRING[32];   { ** Cadena para la variable ** }

  registro   = RECORD        { * Para la Tabla de Simbolos * }
    nombre    : tiponombre;  { * Variable * }
    objeto    : INTEGER;     { * Codigo para Variable * }
    idbloque  : STRING[8];   { * Nivel de Profundidad * }
    atr       : INTEGER;     { * Atributo de Variable * }
    valor     : REAL;        { * Valor de variable * }
  END;

  tipocode = RECORD         { * Para la Tabla de Codigo Intermedio * }
    op, arg1, arg2, arg3 :INTEGER;
  END;

  tipotabwrit = RECORD      { * Tabla para escritura de valores * }
    valor : INTEGER;
    decim : byte;
  END;

VAR   { ** Variables del Analizador Lexicografico ** }
  archivo    : TEXT;            { * Archivo de Codigo Intermedio * }
  c          : CHAR;            { * Caracter leido por SigTok  *   }
  TS         : ARRAY[1..maxts] OF registro;   { * Tabla de Simbolos * }
  PTC        : ARRAY[1..maxcte] OF tipotempo; { * Para Constantes * }
  TE : ARRAY[1..maxerr] OF STRING[50];        { * Tabla Cod. Intermedio *}
  bufferlin  : STRING[255];                   { * Linea leida del Texto *}
  temporal   : tipotempo;                     { * Variable Temporal * }
  n_archivo  : STRING[15];                    { * Nombre del archivo * }
  flag       : BOOLEAN;           { * De caracter leido, para Unget * }
  flagapos   : BOOLEAN;           { * De apostrofe * }
  flagdosp   : BOOLEAN;
  final      : BOOLEAN;           { * De final de archivo * }
  flagtok    : BOOLEAN;           { * De token hallado * }
  numero     : BOOLEAN;           { * De numero * }
  numero1    : BOOLEAN;           { * Auxiliar de numero * }
  flagdecvar : BOOLEAN;           { * Es de declaracion de variables * }
  flagnivel  : BOOLEAN;           { * Es variable de Nivel * }
  reschk     : BOOLEAN;           { * Del analizador sintactico *}
  ini_corr   : byte;              { * Auxiliar del A.Sintactico *}
  numlin, numcol : INTEGER;       { * Numero de linea y columna leida * }
  p,  mtoken  : INTEGER;          { * Numero de caracter leido, token * }
  numerror    : INTEGER;          { * Numero de error * }
  lugar,                          { * Apunta a Tabla de Simbolos * }
  cont_cte,                       { * Numero de Constantes * }
  cont_ident  : INTEGER;          { * Numero de identificadores * }
  estado, Tok : INTEGER;
  valor_real  : REAL;             { * Valor real hallado por lexan *}
```

```
    Vartempo    : Tiponombre;       { * Arreglo temporal * }
    I:INTEGER;                      { * Auxiliar * }
    asig_a_tabla: BOOLEAN;          { * Si se asigna a T.Simbolos *}

      {** Variables del codificador intermedio **}
    cx,
    vtn,
    varx,
    prx,
    ubivarfact : INTEGER;
    TCODE      : ARRAY[1..maxcode] OF tipocode;   {** Codigo Intermedio **}
    pila       : ARRAY[1..maxcode] OF INTEGER;    {** Auxiliar para El TCode*}
    temp2      : STRING[15];

    tabwrit    : ARRAY[1..50] OF tipotabwrit;  { ** Variables a imprimir **}
    idbloque6 : STRING[8];                     { ** Identificador de Nivel *}
    Area_id_bloque : ARRAY [1..16] OF STRING[8];  { * Para ubicar el Nivel *}
    pbloque        : INTEGER;                  { * Numero de Bloques * }
    ininter, ntabwrit : INTEGER;               { * numero de tabla write *}

      {** Variables de pantalla **}
    Origmode, Lastcol, LastRow :word;      { * Modos, columnas y filas * }

      {** Tipos y Variables del Graficador **}

CONST maxvar = 9;           { ** Numero de variables a graficar * }
TYPE tipo_datograf = RECORD   { * De la variable a graficar * }
       nombre    :INTEGER;  { * Ubicacion en Tabla Simbolos *}
       tipo_linea:byte;     { * Tipo de Linea * }
       min, max  :REAL;     { * Valores maximo y minimo * }
       factor    :REAL;     { * Factor de proporciones graficas *}
       x,y       :INTEGER;
       flagrango :BOOLEAN;  { * Si esta en el rango * }
     END;

    tipo_interpola = RECORD
    ntabla, nexpr, nvalor:INTEGER; END;

VAR  nvargraf:byte;               { * numero de variable a graficar *}
    dat_vargraf : ARRAY[1..maxvar] OF tipo_datograf; { Tabla para Graficos *}
    narchwork : STRING[12];      { * Nombre de archivo intermedio *}
    archwork  : TEXT;            { * Tipo de archivo * }
    w_arch    : BOOLEAN;         { * Para escribir en archivo * }
    paso      : INTEGER;         { * De grafico * }
    iniciog, finalg:INTEGER;     { * De grafico * }

    tab_interp : ARRAY[1..maxinterp] OF tipo_interpola;
    ninterp    : INTEGER;

procedure error(numero:integer);
function existe_arch(nombre:string):boolean;
function esalfa(c:char):boolean;
function leecar:char;
procedure ReponeTok;
procedure unget(c:char);
function es_variable(x:integer):boolean;
function comp_cad(cadena1,cadena2:string):boolean;
function instala_cte(cadena:tipotempo):integer;
```

```
function instala_ident(cadena:string):integer;
procedure carga;
function SigTok:integer;

implementation
FUNCTION existe_arch(nombre:STRING):BOOLEAN; { * verifica si existe el archivo * }
   VAR ficheq :FILE;
BEGIN
   Assign(ficheq,nombre);
     {$I-} Reset (ficheq); {$I+}
   IF Ioresult=0 THEN
   BEGIN
      existe_arch:=TRUE;
      Close(ficheq);
   END
   ELSE Existe_arch:=FALSE;
END;

FUNCTION esalfa(c:CHAR):BOOLEAN;    { * Si es 'c' un caracter alfabetico * }
BEGIN
   IF (c IN ['A'..'Z']) OR (c='_') OR (c IN ['0'..'9']) THEN
      esalfa := TRUE
   ELSE esalfa :=FALSE;
END;

PROCEDURE error(numero:INTEGER);      {* Administra el escribe error *}
   VAR i:byte; token:INTEGER;
BEGIN
   gotoxy(1,12);clreol;
   gotoxy(1,13);clreol;
   writeln('ERROR EN LINEA ',numlin,': ',bufferlin);
   FOR i:=0 TO numcol+15 DO write(' '); write('^--');
   write(TE[numero]);
   halt;
END;

FUNCTION leecar:CHAR;   {* Lee caracter desde archivo fuente *}
   VAR d:CHAR;
BEGIN
   IF flag=TRUE THEN leecar:=c
   ELSE
   BEGIN
      IF (numcol > length(bufferlin)) THEN
      BEGIN
         readln(archivo,bufferlin);
         gotoxy(30,12); write('COMPILANDO LINEA :', numlin);
         numcol:=1;
         numlin:=numlin+1;
         d:=' ';
         leecar:=d;
         IF final THEN exit;
      END
      ELSE
      BEGIN
         d:=bufferlin[numcol];
         leecar:=upcase(d);
         numcol:=numcol+1;
      END;
```

```
  END;
  IF EOF(ARCHIVO) THEN FINAL:=TRUE;
  flag:=FALSE;
  IF (d=#3) THEN halt;
END;

PROCEDURE ReponeTok;    { * Simula la Devuelve del Token, cambia flagtok * }
BEGIN
  flagtok:=TRUE;
  mtoken:=tok;
END;

PROCEDURE unget(c:CHAR);   {* Simula devolucion del caracter leido *}
BEGIN
  flag:=TRUE;
END;

FUNCTION es_variable(x:INTEGER):BOOLEAN; {* Si x es variable := true *}
BEGIN
  IF (ts[x].atr = ID_NIVEL) OR (ts[x].atr = ID_FLUJO)
    OR (ts[x].atr = ID_AUX) OR (ts[x].atr = ID_TABLA) THEN
    es_variable:=TRUE
  ELSE es_variable:=FALSE;
END;

FUNCTION comp_cad(cadena1,cadena2:STRING):BOOLEAN;
   {* Compara dos cadenas. Si son iguales := true *}

  VAR j,max:INTEGER;
BEGIN
  j:=1;
  IF length(cadena2) >= length(cadena1) THEN max:=length(cadena2)
  ELSE max:=length(cadena1);
  comp_cad:=FALSE;
  WHILE j <= max DO
  BEGIN
    IF cadena1[j]<>cadena2[j] THEN
    BEGIN
      comp_cad:=FALSE;
      exit;
    END;
    j:=j+1;
  END;
  comp_Cad:=TRUE;
END;

FUNCTION instala_cte(cadena:tipotempo):INTEGER;
  VAR inte:INTEGER;
BEGIN
  inte:=1;
  WHILE inte <= cont_cte DO
  BEGIN
    IF cadena = ptc[inte] THEN
      BEGIN instala_cte:=inte; exit; END;
    inte:=inte+1;
  END;
  cont_cte:=cont_cte+1;
  ptc[cont_cte]:=cadena;
```

```
   instala_Cte:=cont_cte;
END;

FUNCTION parteinic(cad1, cad2:string;t:integer):boolean;
var i, mix : integer;
begin
  cad1:=copy(cad1,1,length(cad1));
  cad2:=copy(cad2,1,length(cad2));
  parteinic:=false;
  if (length(cad1) >= length(cad2)) then mix:=length(cad1)
       else mix := length(cad2);
    if numero1 then if length(cad1) > length(cad2) then mix:=length(cad2)
              else mix := length(cad1);
  if ((ts[t].atr = ID_SUBSIS) and not flagdecvar) then
    begin
       cad2:=copy(cad2,1,length(cad2)-1);
       if cad1 = cad2 then parteinic:=true;
       exit;
    end;
  if ((ts[t].atr = ID_FUNC) and (not flagdecvar)) then
    begin
       if cad1 <= cad2 then parteinic:=true;
       exit;
    end;
  if ((not flagdecvar) and es_variable(t)) then
    begin
       repeat
       if cad1 = cad2 then
          begin
             parteinic:=true; exit;
          end;
       cad1:=copy(cad1,1,length(cad1)-1);
       i:=length(cad1);
       until i < 1;
       parteinic:=false; exit;
    end;
    i:=1;
    while i <= mix do
       begin
            if cad2[i] <> cad1[i] then
             begin
                  parteinic:=false; exit;
             end;
             i:=i+1;
       end;
    parteinic:=true;
end;

FUNCTION instala_ident(cadena:STRING):INTEGER;
  { * Instala el identificador en la Tabla de Simbolos * }

   VAR i:INTEGER;cadenaa: STRING;
BEGIN
   cadenaa:=copy(cadena,1,pos(' ',cadena)-1);
   IF length(cadenaa) = 0 THEN cadenaa:=cadena;
   i:=1;
   WHILE i<=maxpalres DO
   BEGIN
```

```
      IF (ts[i].nombre = cadenaa) THEN
         BEGIN instala_ident:=i; exit; END;
      i:=i+1;
   END;
   i:=cont_ident;
   WHILE i>maxpalres DO
   BEGIN
      IF ((ts[i].nombre = cadenaa) AND (parteinic(idbloqueG,ts[i].idbloque,i)) AND (NOT flagdecvar))
         THEN BEGIN instala_ident:=i; exit; END;
      IF ((ts[i].nombre = cadenaa) AND (numero1)) THEN
         BEGIN instala_ident:=i; exit; END;
      IF (flagdecvar AND (ts[i].nombre = cadenaa) AND (ts[i].idbloque = idbloqueG) AND (NOT numero1))
THEN error(4);
      i:=i-1;
   END;
   IF (NOT flagdecvar) AND (NOT numero1) AND NOT (cadenaa >= '_') THEN error(3);
   cont_ident:=cont_ident+1;
   TS[cont_ident].nombre:=cadenaa;
   TS[cont_ident].idbloque:=idbloqueG;
   instala_ident:=cont_ident;
   IF numero1 THEN
   BEGIN
      ts[cont_ident].valor:=valor_real;
      ts[cont_ident].idbloque:='1';
      ts[cont_ident].atr := id_num;
   END;
   IF cont_ident > maxts THEN error(1);
END;

PROCEDURE carga;     {* Carga los valores iniciales de la Tabla de Simbolos *}
   VAR i:INTEGER;
BEGIN
   TS[1].NOMBRE:='SISTEMA';       TS[1].OBJETO:=tSISTEMA;
   TS[2].NOMBRE:='CONST';         TS[2].OBJETO:=tCONS;
   TS[3].NOMBRE:='VAR';           TS[3].OBJETO:=tVAR;     ts[3].atr:=ID_TIPO;
   TS[4].NOMBRE:='NIVEL';         TS[4].OBJETO:=tNIVEL;   ts[4].atr:=ID_TIPO;
   TS[5].NOMBRE:='FLUJO';         TS[5].OBJETO:=tFLUJO;   ts[5].atr:=ID_TIPO;
   TS[6].NOMBRE:='AUX';           TS[6].OBJETO:=tAUX;     ts[6].atr:=ID_TIPO;
   TS[7].NOMBRE:='TABLA';         TS[7].OBJETO:=tTABLA;   ts[7].atr:=ID_TIPO;
   TS[8].NOMBRE:='CONTROL';       TS[8].OBJETO:=tCONTROL;

   TS[9].NOMBRE:='DT';        TS[9].OBJETO:=tDT;     ts[9].valor:=1; ts[9].atr:=id_CONTROL;
   TS[10].NOMBRE:='DESDE_T';  TS[10].OBJETO:=tDESDE_T;ts[10].valor:=1; ts[10].atr:=id_CONTROL;
   TS[11].NOMBRE:='HASTA_T';  TS[11].OBJETO:=tHASTA_T;ts[11].valor:=1; ts[11].atr:=id_CONTROL;
   TS[12].NOMBRE:='LONGITUD'; TS[12].OBJETO:=tLONGITUD;ts[12].valor:=1;ts[12].atr:=id_CONTROL;
   TS[13].NOMBRE:='RANGOS';       TS[13].OBJETO:=tRANGOS;
   TS[14].NOMBRE:='OBJETIVO';     TS[14].OBJETO:=tOBJETIVO;
   TS[15].NOMBRE:='SUBSISTEMA';   TS[15].OBJETO:=tSUBSIS;
   TS[16].NOMBRE:='FUNCION';      TS[16].OBJETO:=tFUNCION;
   TS[17].NOMBRE:='INICIO';       TS[17].OBJETO:=tINICIO;
   TS[18].NOMBRE:='FIN';          TS[18].OBJETO:=tFIN;
   TS[19].NOMBRE:='SI';           TS[19].OBJETO:=tSI;
   TS[20].NOMBRE:='ENTONCES';     TS[20].OBJETO:=tENTONCES;
   TS[21].NOMBRE:='SINO';         TS[21].OBJETO:=tSINO;
   TS[22].NOMBRE:='MIENTRAS';     TS[22].OBJETO:=tMIENTRAS;
   TS[23].NOMBRE:='HACER';        TS[23].OBJETO:=tHACER;
   TS[24].NOMBRE:='DESDE';        TS[24].OBJETO:=tDESDE;
   TS[25].NOMBRE:='HASTA';        TS[25].OBJETO:=tHASTA;
```

```
TS[26].NOMBRE:='RHASTA';       TS[26].OBJETO:=tRHASTA;
TS[27].NOMBRE:='NO';           TS[27].OBJETO:=tNO;
ts[28].nombre:='O';            TS[28].OBJETO:=tOR;
ts[29].nombre:='Y';            TS[29].OBJETO:=tY;

ts[33].nombre:='PON_GRAFICA'; ts[33].objeto:=tPON_GRAFICA;
TS[34].nombre:='GRAFICA'  ;   TS[34].objeto:=tGRAFICA;     ts[34].atr:=id_func;
ts[35].nombre:='ESCRIBE';     TS[35].OBJETO:=tESCRIBE; TS[35].ATR:=ID_FUNC;
ts[36].nombre:='ESCRIBELN';   TS[36].OBJETO:=tESCRIBELN; TS[36].ATR:=ID_FUNC;
ts[37].nombre:='INTERPOLA';   TS[37].OBJETO:=tINTERPOLA; TS[37].ATR:=ID_FUNC;
ts[38].nombre:='INTERPOLX';   TS[38].OBJETO:=tINTERPOLX; TS[38].ATR:=ID_FUNC;
ts[39].nombre:='TIEMPO'   ;   ts[39].objeto:=tTIEMPO; ts[39].atr:=id_control;
ts[40].nombre:='PI';           ts[40].valor:=pi; ts[40].atr:=id_cons;
TS[41].NOMBRE:='1';            TS[41].valor:=1;  ts[41].atr:=id_num;
ts[42].nombre:='ABS';
ts[44].nombre:='ARCTAN';
ts[46].nombre:='COS';          TS[46].ATR:=ID_FUNC;
ts[48].nombre:='DEMINF';
ts[50].nombre:='DEMORA';
ts[52].nombre:='EXP';
ts[54].nombre:='FRAC';
TS[56].nombre:='ENT';
ts[58].nombre:='CUAD';
ts[62].nombre:='LN';
ts[64].nombre:='LOG';
ts[66].nombre:='RAIZC';
ts[68].nombre:='RND';
ts[70].nombre:='RUIDO';
ts[72].nombre:='SEN';          TS[72].ATR:=ID_FUNC;
i:=42;  WHILE i <= 72 DO BEGIN ts[i].atr:=Id_FUNC; ts[i].valor:=1; i:=i+2; END;
ts[43].nombre:='$$X';            ts[43].atr:=id_var;
ts[45].nombre:='$$X';
ts[47].nombre:='$$X';
ts[49].nombre:='$$X';
ts[51].nombre:='$$X';            ts[51].atr:=id_var;
ts[53].nombre:='$$X';
ts[55].nombre:='$$X';
ts[57].nombre:='$$X';
ts[59].nombre:='$$X';
ts[61].nombre:='$$X';
ts[63].nombre:='$$X';
ts[65].nombre:='$$X';
ts[67].nombre:='$$X';
ts[69].nombre:='$$X';
ts[71].nombre:='$$X';
ts[73].nombre:='$$X';            ts[73].atr:=id_var;
i:=43;  WHILE i <= 73 DO BEGIN ts[i].atr:=tAUX; i:=i+2; END;

FOR i := 1 TO maxpalres DO
   ts[i].idbloque:=idbloqueG; {copia estado de idbloqueg}

  { * Carga los valores de la tabla de Errores * }

TE[1]:='Fuera de Memoria en Tabla de Simbolos';
te[2]:='Esperando Identificador';
te[3]:='Identificador desconocido';
te[4]:='Identificador Duplicado';
te[5]:='Error de Sintaxis';
```

```
te[6]:='Error en Constante Real';
te[7]:='Error en Constante Entera';
te[8]:='Esperando Identificador de Flujo';
te[9]:='Esperando Identificador AUXiliar';
te[10]:='Inesperado Final de Archivo';
te[11]:='Linea muy larga';
te[12]:='Esperando Identificador de Tipo';
te[13]:='Muchos archivos abiertos';
te[14]:='Nombre de archivo Invalido';
te[15]:='Archivo inexistente';
te[16]:='Disco LLeno';
te[17]:='Esperando Identificador de Tabla';
te[18]:='Muchos archivos';
te[19]:='# Datos de tabla diferentes a declarados';
te[20]:='Esperando Identificador de Nivel';
te[21]:='Error en tipo';
te[22]:='# Parametros Diferentes a declarados';
te[23]:='Esperando constante de cadena';
te[24]:='Subsistema o función sin parámetros';
te[25]:='Invalida longitud de cadena';
te[26]:='Incompatibilidad de tipos';
te[27]:='Subrango invalido en tipo base';
te[28]:='Inferior mayor que superior';
te[29]:='Esperando tipo Ordinal';
te[30]:='Esperando entero constante';
te[31]:='Esperando constante';
te[32]:='Esperando Entero o constante real';
te[33]:='Esperando Identificador de Constante';
te[34]:='Resultado de funcion invalida por tipo';
te[35]:='Esperando Identificador de Etiqueta';
te[36]:='Esperando INICIO';
te[37]:='Esperando FIN';
te[38]:='Esperando expresion entera';
te[39]:='Esperando expresion ordinal';
te[40]:='Esperando expresion Booleana o logica';
te[41]:='Tipos de operandos son incompatibles con el operador';
te[42]:='Error en expresion';
te[43]:='Asignacion ilegal';
te[44]:='Idenficador de campo esperado';
te[45]:='Archivo muy grande';
te[46]:='Esperando identificador de control';
te[47]:='Esperando HACER';
te[48]:='Esperando ENTONCES';
te[49]:='Esperando HASTA o RHASTA';
te[50]:='Division por cero';
te[51]:='Tipo de archivo no valido';
te[52]:='Esperando variable de cadena';
te[53]:='Esperando expresion de cadenas';
te[54]:='Disponible                    ';
te[55]:='Constante fuera del rango';
te[56]:='Esperando variable de archivo';
te[57]:='Esperando expresion entera o real';
te[58]:='Etiqueta no se encuentra en el bloque actual';
te[59]:='Error en ejecucion: Valor fuera de Rango.';
te[60]:='Esperando ";"';
te[61]:='Esperando ":"';
te[62]:='Esperando ","';
te[63]:='Esperando "("';
```

```
   te[64]:='Esperando ")"';
   te[65]:='Esperando "="';
   te[66]:='Esperando ":="';
   te[67]:='Esperando "["';
   te[68]:='Esperando "]"';
   te[69]:='Esperando "."';
   te[70]:='Esperando ".."';
   te[71]:='Muchas variables declaradas';
   te[72]:='Variable de Control de DESDE-HASTA invalido';
   te[73]:='Esperando variable entera';
   te[74]:='Esperando variable ordinal';
   te[75]:='Esperando expresion ';
   te[76]:='No encuentro memoria para correr programa';
   te[77]:='Aborta Compilacion ';
   te[78]:='Fuera de Memoria en Tabla de Codigo.';
   te[79]:='Error en declaracion';
   te[80]:='Falta <SISTEMA>';
   TE[81]:='Falta Nombre de SISTEMA';
   te[82]:='Error al tomar constante real';
END;

FUNCTION SigTok:INTEGER;
  { * Analizador Sintactico * }
  { * Responde con un Token * }

   VAR tkv:INTEGER;
   FUNCTION lexan:INTEGER;
   { * Analizador de una cadena leida * }
   BEGIN
      WHILE TRUE DO
      BEGIN
         c:=leecar;
         IF (c <> ' ')  THEN
         BEGIN
            IF flagdosp THEN
            BEGIN
               unget(c);
               Lexan:=DOS_PUNTOS_H; flagdosp:=FALSE; exit;
            END;
            IF (c IN ['0'..'9']) AND (flagapos) THEN
            BEGIN
               p:=0;
               REPEAT
                  p:=p+1;
                  temporal[p]:=c;
                  c:=leecar;
               UNTIL NOT (c IN ['0'..'9']);
               IF c <> '.' THEN
                  IF p>9 THEN error(55)
                  ELSE
                  BEGIN
                     unget(c);
                     numero1:=TRUE;
                     VARTEMPO:=TEMPORAL;
                     val(COPY(VARTEMPO,1,POS(' ',VARTEMPO)-1),VALOR_real,ESTADO);
                     IF estado <> 0 THEN error(82);
                     lugar:=instala_ident(temporal);
                     Lexan:=NUM_R;
```

```
            ts[lugar].valor:=valor_real;
            numero1:=FALSE;
            exit;
         END;
            {si es punto }
      IF C = '.' THEN
      BEGIN
         c:=leecar;
         IF (c='.' ) THEN
         BEGIN
            numero1:=TRUE;
            VARTEMPO:=TEMPORAL;
            val(COPY(VARTEMPO,1,POS(' ',VARTEMPO)-1),VALOR_REAL,ESTADO);
            IF estado <> 0 THEN error(82);
            lugar:=instala_ident(temporal);
            Lexan:=NUM_R;
            ts[lugar].valor:=valor_REAL;
            flagdosp:=TRUE;
            numero1:=FALSE;
            EXIT;
         END;
         IF c IN  ['0'..'9'] THEN
         BEGIN
            p:=p+1;
            temporal[p]:='.';
         END;
         WHILE (c IN ['0'..'9']) DO
         BEGIN
            p:=p+1;
            temporal[p]:=c;
            c:=leecar;
         END;
         unget(c);
         numero1:=TRUE;
         VARTEMPO:=TEMPORAL;
         val(COPY(VARTEMPO,1,POS(' ',VARTEMPO)-1),VALOR_real,ESTADO);
         IF estado <> 0 THEN error(82);
         lugar:=instala_ident(temporal);
         Lexan:=NUM_R;
         ts[lugar].valor:=valor_REAL;
         numero1:=FALSE;
         exit;
      END; ( * fin de si es punto *)
   END;
   IF esalfa(c) AND (flagapos) THEN
   BEGIN
      p:=0;
      REPEAT
         p:=p+1;
         temporal[p]:=c;
         c:=leecar;
      UNTIL NOT esalfa(c);
      unget(c);
      lugar:=instala_ident(temporal);
      IF (lugar <= 42) THEN
      BEGIN
         Lexan:=ts[lugar].objeto;
         exit;
```

```
    END
    ELSE
    BEGIN
       Lexan:=IDENT;
       exit;
    END
END;
IF c IN ['*', '/', '-','+','[',']', ',', ';', '=','(',')','^'] THEN
BEGIN
   IF c = '(' THEN
   BEGIN
      c:=leecar;
      IF c='*' THEN
      BEGIN
         WHILE c<> ')' DO
         BEGIN
            REPEAT
               c:=leecar;
            UNTIL (c='*');
            c:=leecar;
            IF c <> ')' THEN unget(c);
         END;
      END
      ELSE
      BEGIN
         unget(c);
         Lexan:=PAR_I;
         exit;
      END;
   END
   ELSE
   BEGIN
      Lexan:=ord(c);
      exit;
   END;
END;
IF c=':' THEN
BEGIN
   c:=leecar;
   IF c='=' THEN BEGIN Lexan:=ASIG; exit;  END
   ELSE
      unget(c);
   Lexan:=DOS_PUNTOS_V;
   Exit;
END;
IF c='.' THEN
BEGIN
   c:=leecar;
   IF c='.' THEN BEGIN Lexan:=DOS_PUNTOS_H; exit; END
   ELSE unget(c);
   Lexan:=PUNTO;
   exit;
END;
IF c='<' THEN
BEGIN
   c:=leecar;
   IF c='=' THEN BEGIN Lexan:=MENOR_IGUAL; exit; END;
   IF c='>' THEN BEGIN Lexan:=DISTINTO; exit; END;
```

```
            unget(c);
            Lexan:=MENOR;
            exit;
          END;
          IF c='>' THEN
          BEGIN
            c:=leecar;
            IF c='=' THEN BEGIN Lexan:=MAYOR_IGUAL; exit; END;
            unget(c);
            Lexan:=MAYOR;
            exit;
          END;
          IF c='{' THEN
          BEGIN
            REPEAT
              c:=leecar;
            UNTIL (c='}');
            c:=leecar;
          END;
          IF (NOT esalfa(c) AND (c <> ' '))  THEN error(5);
          unget(c);
        END;
         {fin del if <> blanco }
      END; {while}
   END; { ** final de lexan **}

BEGIN {** inicio de sigtok **}
   fillchar(temporal,sizeof(temporal),' ');
   IF flagtok THEN
   BEGIN
      SigTok:=Mtoken;
      flagtok:=FALSE;
      exit;
   END
   ELSE
   BEGIN
      tkv:=lexan;
      sigtok:=tkv;
      mtoken:=tkv;
   END;
END;

BEGIN
END.
```