EXEC_TimeStamp_2.pas

{ freePascal 2.6.2 (entorno BloodShed v1.9) }
*(*Este programa simula la ejecución de los planes de acción que hay en el fichero de texto especificado.*
En cada línea del fichero se especifica un plan de ejecución <planEjec> *con la siguiente sintaxis:*
<planEjec> ::= 'P' <numplan> ':' {<opPlan>} *y* <opPlan> ::= <idOp> <num> ['(' <idDato> ')'] ';'
donde <numplan> y <num> *son números naturales,* <idDato> *una letra mayúscula, y* <idOp> la letra que representa la operación a ejecutar:
***A** (abortar),* ***C** (validar),* ***P** (iniPlan),* ***T** (iniTrans),* ***R** (leer) o* ***W** (escribir)*
Si no se le pasa como parámetro de la ejecución ningún fichero, el programa nuestra un cuadro de diálogo para la selección del fichero.

Santiago Velilla, 3 Junio 2013
Dpto. Informática e Ingeniería de Sistemas, Univ. Zaragoza
**)*

```
PROGRAM EXEC_Plan_TimeStamp_1(input, output);
USES UtilFich;

const MAX_TRANS = 10;  {número máximo de transacciones en un plan de ejecución}
      ch_EOLN = chr(13);   ch_TAB = chr(9);

type tpNombre = string[80];                    { tipo para el nombre del fichero }
     tpAccion = (validar, abortar, leer, escribir, iniTrans, iniPlan, finPlan, ilegal);
     tpEstadoTransacc = (inactiva, activa, pendValidar, validada, abortada);
     tpOpPlan = record
                  accion: tpAccion;
                  idElem: integer;    {número de la transacción o del plan }
                  idDato:   char;     {sólo para acciones de leer y escribir}
                end;
     tpInfoTrans = record
                     estado:  tpEstadoTransacc;
                     TStamp:  integer
                   end;
     tpInfoDato  = record
                     TS_lect, TS_escr:       integer;
                     Trans_lect, Trans_escr: integer
                   end;

var  fPlan, fLst: text;    { fichero de texto con el plan, y fichero de listado }
     nombFichPlan, nombFichLst : tpNombre;     { nombres de los ficheros usados }
     tbl_Trans: array [1..MAX_TRANS] of tpInfoTrans;
     tbl_Datos: array ['A'..'Z'] of tpInfoDato;
     TimeStamp: integer; {contador que simula los valores de las marcas de tiempo}
     operacion: tpOpPlan;
```

```
function mayuscula(c: char): char;     { devuelve c en mayúsculas }
begin
  if c in ['a'..'z'] then mayuscula:=chr(ord(c)-ord('a')+ord('A')) else mayuscula:=c
end;
```

```
function ValDig(c: char): integer;     { devuelve el dígito correspondiente a c }
begin
  ValDig := ord(c) - ord('0')
end;
```

```
function esDigito(c: char): boolean;          { devuelve true sii c es un dígito }
begin
  esDigito := (c <= '9') and (c >= '0')
end;
```

```
procedure leerCar(var c: char);
begin
  if eoln(fPlan) then c := ch_EOLN else read(fPlan, c);  c:=mayuscula(c)
end;
```

```
procedure leerNatural(var c: char;  var num: integer);
begin
  num := 0;
  while esDigito(c) do begin { añadir un nuevo dígito }
    num:= num * 10 + ValDig(c);      leerCar(c)
  end    {; writeln(num);}
end;
```

```
procedure writeMsg(msg: string);  { muestra un mensaje por pantalla y fichero de listado }
begin
  write(msg); write(fLst, msg)
end;
```

```
procedure writeLnMsg(msg: string);  { muestra un mensaje por pantalla y fichero de listado y pasa a la siguiente línea}
begin
  writeln(msg); writeln(fLst, msg)
end;
```

```
function next_TimeStamp: integer;         { devuelve true sii c es un dígito }
begin
  TimeStamp := TimeStamp + 1;    next_TimeStamp := TimeStamp
end;
```

```
procedure writeOpPlan(opPlan: tpOpPlan);  { muestra la acción del plan }
var msg: string;
begin with opPlan do begin
  case accion of
   abortar : writeStr(msg, 'Abortar T', idElem:1);
   validar : writeStr(msg, 'Validar T', idElem:1);
   iniPlan : writeStr(msg, 'Iniciar P', idElem:1);
   iniTrans: writeStr(msg, 'Iniciar T', idElem:1);
   leer    : writeStr(msg, 'T', idElem:1, '_Read(', idDato, ')');
   escribir: writeStr(msg, 'T', idElem:1, '_Write(', idDato, ')');
   finPlan : writeStr(msg, 'fin Plan');
   ilegal  : writeStr(msg, 'operación ilegal')
  end;    writeMsg(msg)
end end;
```

```
procedure IniciarPlan(numPlan: integer);  { iniciar marcas de tiempo y variables }
var i: integer;   c: char;  msg: string;
begin
  for i:=1 to MAX_TRANS do with tbl_Trans[i] do begin {iniciar transacción i-ésima}
    estado:= inactiva; TStamp:= -1
  end;
  for c:='A' to 'Z' do with tbl_Datos[c] do begin {iniciar dato c-ésimo}
    TS_lect:=0;  TS_escr:=0;  Trans_lect:=0;  Trans_escr:=0
  end;  writeStr(msg, 'plan P', numPlan:1, ' iniciado');  writeLnMsg(msg)
end;
```

```
procedure Lectura_transacc(id_Trans: integer;  id_Dato: char);  { }
var msg: string;
begin with tbl_Trans[id_Trans], tbl_Datos[id_Dato] do begin
  if estado = activa then begin
    if TS_escr > TStamp then begin estado := abortada;
      writeStr(msg, 'T', id_Trans:1, ' abortada al leer"', id_Dato, '" (TS_lect > TStamp)');  writeMsg(msg)
    end else begin
      if TS_lect < TStamp then begin TS_lect := TStamp;  Trans_lect := id_Trans end;
      writeStr(msg, 'T', id_Trans:1, ' lee de elemento "', id_Dato, '"'); writeMsg(msg)
    end;
    writeStr(msg,  ch_TAB, 'TStamp T', id_Trans:1, '=', TStamp:1, ' TS_lect_', id_Dato, '=', TS_lect:1, ' TS_escr_', id_Dato, '=', TS_escr:1);
    writeLnMsg(msg)
  end else writeLnMsg('ERR: transaccion no en estado activo')
end end;
```

```
procedure Escritura_transacc(id_Trans: integer;  id_Dato: char);  { }
var msg: string;
begin with tbl_Trans[id_Trans], tbl_Datos[id_Dato] do begin
  if estado = activa then begin
    if TS_lect > TStamp then begin estado := abortada;
      writeStr(msg,  'T', id_Trans:1, ' abortada al leer "', id_Dato, '" (TS_lect > TStamp)');  writeMsg(msg)
    end else if TS_escr > TStamp then begin {se aplica regla de Thomas}
      writeStr(msg,  'T', id_Trans:1, ' no escribe (TS_escr > TStamp)'); writeMsg(msg)
    end else begin {escribe}
      TS_escr := TStamp;  Trans_escr := id_Trans;
      writeStr(msg,  'T', id_Trans:1, ' escribe en "', id_Dato, '"'); writeMsg(msg)
    end;
    writeStr(msg,  ch_TAB, 'TStamp T', id_Trans:1, '=', TStamp:1, ' TS_lect_', id_Dato, '=', TS_lect:1, ' TS_escr_', id_Dato, '=', TS_escr:1);
    writeLnMsg(msg)
  end else writeLnMsg('ERR: transaccion no en estado activo')
end end;
```

```
procedure IniciarTransacc(numTrans: integer);  { iniciar marcas de tiempo y estado }
begin with tbl_Trans[numTrans] do begin
  if estado = inactiva then begin
    estado := activa;  TStamp := next_TimeStamp; writeLnMsg('iniciada')
  end else writeLnMsg('ERR: ya estaba activa')
end end;
```

```
procedure AbortarTransacc(numTrans: integer);
begin with tbl_Trans[numTrans] do begin
  if estado = activa then begin
    estado := abortada;   writeLnMsg('abortada')
  end else writeLnMsg('ERR: no estaba activa')
end end;
```

```
procedure ValidarTransacc(numTrans: integer);
begin with tbl_Trans[numTrans] do begin
  if (estado = activa) or (estado = pendValidar) then begin
    estado := validada;  writeLnMsg('validada')
  end else writeLnMsg('ERR: no estaba activa')
end end;
```

```
procedure ejecutarOpPlan(opPlan: tpOpPlan);  { muestra la acción del plan }
begin with opPlan do begin
  writeOpPlan(opPlan); writeMsg(ch_TAB);
  case accion of
   abortar : AbortarTransacc(idElem);
   validar : ValidarTransacc(idElem);
   iniPlan : IniciarPlan(idElem);
   iniTrans: IniciarTransacc(idElem);
   leer    : Lectura_transacc(idElem, idDato);
   escribir: Escritura_transacc(idElem, idDato);
   otherwise
  end
end end;
```

```
procedure leerOpPlan(var opPlan: tpOpPlan);  { obtener la operación a ejecutar }
var c: char;
begin with opPlan do begin
  repeat leerCar(c) until c<>' ';
  case c of
   'A' : accion := abortar;
   'C' : accion := validar;
   'P' : accion := iniPlan;
   'T' : accion := iniTrans;
   'R' : accion := leer;
   'W' : accion := escribir;
   ch_EOLN: accion := finPlan;
   otherwise accion := ilegal
  end;
  if accion <> ilegal then begin
   if accion <> finPlan then begin
    leerCar(c);   leerNatural(c, idElem);  idDato:=' ';
    if idElem > 0 then begin
      if (accion = leer) or (accion = escribir) then begin
        while c = ' ' do leerCar(c);
        if c = '(' then begin
          repeat leerCar(c) until c<>' ';
          if c in ['A'..'Z'] then begin
            idDato:=c;   repeat leerCar(c) until c<>' ';
            if c <> ')' then writeMsg('ERR: se esperaba un "("')
          end else writeMsg('ERR: se esperaba una letra mayúscula')
        end else writeMsg('ERR: se esperaba un ")"')
      end
    end else writeMsg('ERR: se esperaba un número > 0')
   end
  end else writeMsg('ERR: operación ilegal');
  while (c<>';') and (c<>':') and (c<>ch_EOLN) do leerCar(c)
end end;
```

```
procedure seleccionarFicheros(var nombFichPlan, nombFichLst: string);
var  path, nam, ext: string;      hayFich: boolean;
begin
  if paramCount > 0 then begin  { el nombre del fichero es parámetro de la ejecución }
    nombFichPlan := paramStr(1);  hayFich := existeFichero(nombFichPlan)
  end else hayFich := selFichLect(nombFichPlan);
  if hayFich then begin { existe el fichero }
    splitNamFich(nombFichPlan, path, nam, ext);    nombFichLst:=path+nam+'.lst';
    if paramCount > 0 then { el nombre del fichero de listado es parámetro }
      if paramCount > 1 then nombFichLst := paramStr(2) else
    else hayFich := selFichEscr(nombFichLst);
    if not hayFich then nombFichLst := ''
  end {; writeln('fich.Plan = ''', nombFichPlan, '''  fich.Lst = ''', nombFichLst, '''')}
end;
```

```
begin  { del programa principal }
  seleccionarFicheros(nombFichPlan, nombFichLst);
  if nombFichPlan <> '' then begin { existe el fichero con el plan}
    assign(fPlan, nombFichPlan); reset(fPlan);      {preparar fichero para leer }
    assign(fLst, nombFichLst);   rewrite(fLst); {preparar fichero para escribir}
    TimeStamp:=0;
    while not eof(fPlan) do begin       { procesar una línea del fichero (un plan)}
      leerOpPlan(operacion);
      if operacion.accion = iniPlan then begin
        repeat
          ejecutarOpPlan(operacion);    leerOpPlan(operacion)
        until operacion.accion = finPlan;
        writeLnMsg('fin Plan')
      end else writeMsg('ERR: se esperaba accion de inicio de un plan');
      readln(fPlan);  writeLnMsg('');  writeLnMsg(''); { pasar a siguiente línea }
    end;
    close(fPlan);  if nombFichLst <> '' then close(fLst);{ disociar los ficheros }
  end;
  writeln;  write('Pulsar <CR> para terminar'); readln  { esperar confirmación }
end.
```

`Iniciar P1` plan P1 iniciado `Iniciar T1` iniciada `Iniciar T2` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=1 TS_lect_X=1 TS_escr_X=0 `T2_Read(X)` T2 lee de elemento "X" TStamp T2=2 TS_lect_X=2 TS_escr_X=0 `T1_Write(X)` T1 abortada al leer "X" (TS_lect > TStamp) TStamp T1=1 TS_lect_X=2 TS_escr_X=0 `T2_Write(X)` T2 escribe en "X" TStamp T2=2 TS_lect_X=2 TS_escr_X=2 `Validar T2` validada `Validar T1` ERR: no estaba activa `fin Plan`	`Iniciar P3` plan iniciado `Iniciar T1` iniciada `Iniciar T2` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=5 TS_lect_X=5 TS_escr_X=0 `T2_Write(X)` T2 escribe en "X" TStamp T2=6 TS_lect_X=5 TS_escr_X=6 `T1_Write(X)` T1 no escribe (TS_escr > TStamp) TStamp T1=5 TS_lect_X=5 TS_escr_X=6 `Validar T2` validada `Validar T1` validada `fin Plan`
`Iniciar P2` plan iniciado `Iniciar T1` iniciada `Iniciar T2` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=3 TS_lect_X=3 TS_escr_X=0 `T2_Read(X)` T2 lee de elemento "X" TStamp T2=4 TS_lect_X=4 TS_escr_X=0 `T2_Write(X)` T2 escribe en "X" TStamp T2=4 TS_lect_X=4 TS_escr_X=4 `T1_Write(X)` T1 abortada al leer "X" (TS_lect > TStamp) TStamp T1=3 TS_lect_X=4 TS_escr_X=4 `Validar T1` ERR: no estaba activa `Validar T2` validada `fin Plan`	`Iniciar P4` plan iniciado `Iniciar T1` iniciada `Iniciar T2` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=7 TS_lect_X=7 TS_escr_X=0 `T1_Write(X)` T1 escribe en "X" TStamp T1=7 TS_lect_X=7 TS_escr_X=7 `T2_Read(X)` T2 lee de elemento "X" TStamp T2=8 TS_lect_X=8 TS_escr_X=7 `T2_Write(X)` T2 escribe en "X" TStamp T2=8 TS_lect_X=8 TS_escr_X=8 `Validar T2` validada `Validar T1` validada `fin Plan`

`Iniciar P1` plan P1 iniciado `Iniciar T2` iniciada `Iniciar T1` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=2 TS_lect_X=2 TS_escr_X=0 `T2_Read(X)` T2 lee de elemento "X" TStamp T2=1 TS_lect_X=2 TS_escr_X=0 `T1_Write(X)` T1 escribe en "X" TStamp T1=2 TS_lect_X=2 TS_escr_X=2 `T2_Write(X)` T2 abortada al leer "X" (TS_lect > TStamp) TStamp T2=1 TS_lect_X=2 TS_escr_X=2 `Validar T2` ERR: no estaba activa `Validar T1` validada `fin Plan`	`Iniciar P3` plan P3 iniciado `Iniciar T2` iniciada `Iniciar T1` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=6 TS_lect_X=6 TS_escr_X=0 `T2_Write(X)` T2 abortada al leer "X" (TS_lect > TStamp) TStamp T2=5 TS_lect_X=6 TS_escr_X=0 `T1_Write(X)` T1 escribe en "X" TStamp T1=6 TS_lect_X=6 TS_escr_X=6 `Validar T2` ERR: no estaba activa `Validar T1` validada `fin Plan`
`Iniciar P2` plan P2 iniciado `Iniciar T2` iniciada `Iniciar T1` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=4 TS_lect_X=4 TS_escr_X=0 `T2_Read(X)` T2 lee de elemento "X" TStamp T2=3 TS_lect_X=4 TS_escr_X=0 `T2_Write(X)` T2 abortada al leer "X" (TS_lect > TStamp) TStamp T2=3 TS_lect_X=4 TS_escr_X=0 `T1_Write(X)` T1 escribe en "X" TStamp T1=4 TS_lect_X=4 TS_escr_X=4 `Validar T1` validada `Validar T2` ERR: no estaba activa `fin Plan`	`Iniciar P4` plan P4 iniciado `Iniciar T2` iniciada `Iniciar T1` iniciada `T1_Read(X)` T1 lee de elemento "X" TStamp T1=8 TS_lect_X=8 TS_escr_X=0 `T1_Write(X)` T1 escribe en "X" TStamp T1=8 TS_lect_X=8 TS_escr_X=8 `T2_Read(X)` T2 abortada al leer"X" (TS_lect > TStamp) TStamp T2=7 TS_lect_X=8 TS_escr_X=8 `T2_Write(X)` ERR: transaccion no en estado activo `Validar T2` ERR: no estaba activa `Validar T1` validada `fin Plan`