```
Resultado de la ejecución de los planes del fichero "ej_plan_3a.txt":

P1; T1; T2; R1(X); R2(X); W1(X); W2(X); C2; C1;
P2; T1; T2; R1(X); R2(X); W2(X); W1(X); C1; C2;
P3; T1; T2; R1(X); W2(X); R2(x); W1(X); C2; C1;
P4; T1; T2; R1(X); W2(X); W1(X); R2(x); C2; C1;
P5; T1; T2; R1(X); W1(X); R2(X); W2(X); C2; C1;
P6; T1; T2; R1(X); W1(X); W2(X); R2(X); C2; C1;
```

Iniciar P1 comienza plan P1 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 escribir "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 ERROR: transaccion abortada fin Plan P1 plan P1 finalizado	Iniciar P4 comienza plan P4 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 T2_Write(X) T2 escribe en "X" TStamp T2=8 TS_lect_X=7 TS_escr_X=8 T1_Write(X) T1 NO ESCRIBE (TS_escr > TStamp y Thomas) TStamp T1=7 TS_lect_X=7 TS_escr_X=8 T2_Read(X) T2 lee de elemento "X" TStamp T2=8 TS_lect_X=8 TS_escr_X=8 Validar T2 validada Validar T1 validada fin Plan P4 plan P4 finalizado
Iniciar P2 comienza plan P2 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 escribir "X" (TS_lect > TStamp) TStamp T1=3 TS_lect_X=4 TS_escr_X=4 Validar T1 ERROR: transaccion abortada Validar T2 validada fin Plan P2 plan P2 finalizado	Iniciar P5 comienza plan P5 Iniciar T1 iniciada Iniciar T2 iniciada T1_Read(X) T1 lee de elemento "X" TStamp T1=9 TS_lect_X=9 TS_escr_X=0 T1_Write(X) T1 escribe en "X" TStamp T1=9 TS_lect_X=9 TS_escr_X=9 T2_Read(X) T2 lee de elemento "X" TStamp T2=10 TS_lect_X=10 TS_escr_X=9 T2_Write(X) T2 escribe en "X" TStamp T2=10 TS_lect_X=10 TS_escr_X=10 Validar T2 validada Validar T1 validada fin Plan P5 plan P5 finalizado
Iniciar P3 comienza plan P3 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 T2_Read(X) T2 lee de elemento "X" TStamp T2=6 TS_lect_X=6 TS_escr_X=6 T1_Write(X) T1 abortada al escribir "X" (TS_lect > TStamp) TStamp T1=5 TS_lect_X=6 TS_escr_X=6 Validar T2 validada Validar T1 ERROR: transaccion abortada fin Plan P3 plan P3 finalizado	Iniciar P6 comienza plan P6 Iniciar T1 iniciada Iniciar T2 iniciada T1_Read(X) T1 lee de elemento "X" TStamp T1=11 TS_lect_X=11 TS_escr_X=0 T1_Write(X) T1 escribe en "X" TStamp T1=11 TS_lect_X=11 TS_escr_X=11 T2_Write(X) T2 escribe en "X" TStamp T2=12 TS_lect_X=11 TS_escr_X=12 T2_Read(X) T2 lee de elemento "X" TStamp T2=12 TS_lect_X=12 TS_escr_X=12 Validar T2 validada Validar T1 validada fin Plan P6 plan P6 finalizado

```
Resultado de la ejecución de los planes del fichero "ej_plan_3b.txt":

P1; T2; T1; R1(X); R2(X); W1(X); W2(X); C2; C1;
P2; T2; T1; R1(X); R2(X); W2(X); W1(X); C1; C2;
P3; T2; T1; R1(X); W2(X); R2(x); W1(X); C2; C1;
P4; T2; T1; R1(X); W2(X); W1(X); R2(x); C2; C1;
P5; T2; T1; R1(X); W1(X); R2(X); W2(X); C2; C1;
P6; T2; T1; R1(X); W1(X); W2(X); R2(X); C2; C1;
```

Iniciar P1 comienza plan P1 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 escribir "X" (TS_lect > TStamp) TStamp T2=1 TS_lect_X=2 TS_escr_X=2 Validar T2 ERROR: transaccion abortada Validar T1 validada fin Plan P1 plan P1 finalizado	Iniciar P4 comienza plan P4 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 T2_Write(X) T2 abortada al escribir "X" (TS_lect > TStamp) TStamp T2=7 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) ERROR: transaccion abortada Validar T2 ERROR: transaccion abortada Validar T1 validada fin Plan P4 plan P4 finalizado
Iniciar P2 comienza plan P2 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 escribir "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 ERROR: transaccion abortada fin Plan P2 plan P2 finalizado	Iniciar P5 comienza plan P5 Iniciar T2 iniciada Iniciar T1 iniciada T1_Read(X) T1 lee de elemento "X" TStamp T1=10 TS_lect_X=10 TS_escr_X=0 T1_Write(X) T1 escribe en "X" TStamp T1=10 TS_lect_X=10 TS_escr_X=10 T2_Read(X) T2 abortada al leer"X" (TS_escr > TStamp) TStamp T2=9 TS_lect_X=10 TS_escr_X=10 T2_Write(X) ERROR: transaccion abortada Validar T2 ERROR: transaccion abortada Validar T1 validada fin Plan P5 plan P5 finalizado
Iniciar P3 comienza plan P3 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 escribir "X" (TS_lect > TStamp) TStamp T2=5 TS_lect_X=6 TS_escr_X=0 T2_Read(X) ERROR: transaccion abortada T1_Write(X) T1 escribe en "X" TStamp T1=6 TS_lect_X=6 TS_escr_X=6 Validar T2 ERROR: transaccion abortada Validar T1 validada fin Plan P3 plan P3 finalizado	Iniciar P6 comienza plan P6 Iniciar T2 iniciada Iniciar T1 iniciada T1_Read(X) T1 lee de elemento "X" TStamp T1=12 TS_lect_X=12 TS_escr_X=0 T1_Write(X) T1 escribe en "X" TStamp T1=12 TS_lect_X=12 TS_escr_X=12 T2_Write(X) T2 abortada al escribir "X" (TS_lect > TStamp) TStamp T2=11 TS_lect_X=12 TS_escr_X=12 T2_Read(X) ERROR: transaccion abortada Validar T2 ERROR: transaccion abortada Validar T1 validada fin Plan P6 plan P6 finalizado

EXEC_TimeStamp.pas

{ freePascal 2.6.4 (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.
Su interés es únicamente didáctico, para ilustrar el funcionamiento de la técnica del Time-Stamping en el procesamiento de transacciones.

Santiago Velilla, 21 Mayo 2014
Dpto. Informática e Ingeniería de Sistemas, Univ. Zaragoza
**)*

```
PROGRAM EXEC_Plan_TimeStamp(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}
     planActivo: 0..MAXINT;    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 num:= num * 10 + ValDig(c);  leerCar(c) end    {; writeln(num);}
end;
```

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

```
procedure writeLnMsg(msg: string);  { muestra un mensaje por pantalla y en el fichero de listado, en una línea}
 begin writeln(msg); writeln(fLst, msg) end;
```

```
procedure writeERR_estado(st:  tpEstadoTransacc);  { muestra un mensaje de error junto al estado de la transacción}
var msg: string;
begin
  writeStr(msg, 'ERROR: transaccion ', st);   writeLnMsg(msg)
end;
```

```
function next_TimeStamp: integer;          { devuelve la marca de tiempo actual }
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 P', planActivo:1);
   ilegal  : writeStr(msg, 'operación ilegal')
  end;    writeMsg(msg)
end end;
```

```
procedure FinalizarPlan;  { operaciones para terminar el plan }
var msg: string;
begin
  writeStr(msg, 'plan P', planActivo:1, ' finalizado');  writeLnMsg(msg);
  planActivo:=-1
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, 'comienza plan P', numPlan:1);  writeLnMsg(msg);
  planActivo:=numPlan
end;
```

```
procedure Lectura_transacc(id_Trans: integer;  id_Dato: char); {ejecuta la lectura de un dato por la transacc.}
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_escr > 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 writeERR_estado(estado)
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 escribir "', 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 y Thomas)'); 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 writeERR_estado(estado)
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 writeERR_estado(estado)
end end;
```

```
procedure AbortarTransacc(numTrans: integer);
begin with tbl_Trans[numTrans] do begin
  if estado = activa then begin
    estado := abortada;   writeLnMsg('abortada')
  end else writeERR_estado(estado)
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 writeERR_estado(estado)
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);
   finPlan : FinalizarPlan;
   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;   planActivo:=-1;
    while not eof(fPlan) do begin          { procesar una línea del fichero (un plan)}
      if not eoln(fPlan) then begin
        leerOpPlan(operacion);
        if operacion.accion = iniPlan then begin {ejecutar el plan de acción}
          ejecutarOpPlan(operacion);
          repeat
            leerOpPlan(operacion);  ejecutarOpPlan(operacion)
          until operacion.accion = finPlan;
        end else writeMsg('ERR: se esperaba accion de inicio de un plan');
        readln(fPlan);  writeLnMsg('');      { pasar a siguiente línea }
        writeLnMsg('-----------------------------------------------------------------');
        writeLnMsg('');   writeLnMsg('')
      end else begin writeLnMsg('');  readln(fPlan) end
    end;
    close(fPlan);  if nombFichLst <> '' then close(fLst);{ disociar los ficheros }
  end;
  writeln;  write('Pulsar <CR> para terminar'); readln  { esperar confirmación }
end.
```