1 Cuestión 1

	zona estática				zona dinámica				pantalla
	c1	c2	pc	qc	vd1	vd2	vd3	vd4	
1	?	?	?	?					
2	0	2	?	?					
3	0	2	vd1	?	?				
4	0	2	vd1	?	0				
5	0	2	vd2	?	(0)	?			
6	0	2	vd2	?	(0)	2			
7	0	2	vd2	?	(0)	2			2 0
8	0	2	vd2	vd2	(0)	2			
9	0	2	vd2	vd2	(0)	0			
10	0	2	vd2	vd2	(0)	0			0 2
11	0	2	vd2	vd3	(0)	0	?		
12	0	2	vd2	vd3	(0)	0	2		
13	0	2	?	vd3	(0)	-	2		
14	0	2	vd4	vd3	(0)	-	2	?	
15	0	2	vd4	vd3	(0)	-	2	0	
16	0	2	vd3	vd3	(0)	-	2	(0)	
17	0	2	vd3	vd3	(0)	-	2	(0)	: 7 / 2

	zona estática				zona dinámica			pantalla
	c1	c2	pc	qc	vd1	vd2/4	vd3	
1	?	?	?	?				
2	0	2	?	?				
3	0	2	vd1	?	?			
4	0	2	vd1	?	0			
5	0	2	vd2	?	(0)	?		
6	0	2	vd2	?	(0)	2		
7	0	2	vd2	?	(0)	2		2 0
8	0	2	vd2	vd2	(0)	2		
9	0	2	vd2	vd2	(0)	0		
10	0	2	vd2	vd2	(0)	0		0 2
11	0	2	vd2	vd3	(0)	0	?	
12	0	2	vd2	vd3	(0)	0	2	
13	0	2	?	vd3	(0)	-	2	
14	0	2	vd4	vd3	(0)	?	2	
15	0	2	vd4	vd3	(0)	0	2	
16	0	2	vd3	vd3	(0)	(0)	2	
17	0	2	vd3	vd3	(0)	(0)	2	: 7 / 2

`c1` y `c2` deben ser de tipo numérico (real o entero). `p` y `q` del mismo tipo, puntero a real o a entero. Los valores entre paréntesis se refieren a zonas perdidas. Las acciones no recomendables son precisamente las que provocan pérdida de memoria: en las líneas 4 y 15 se pierden las variables apuntadas por `pc` al hacer un `new` sin retener la posición de la variable apuntada.

2 Cuestión 2

```
type Tyele = ini .. fin;
     Tyconj = set of Tyele;

procedure quitaExtremos (var c: Tyconj);
(* Entrada-salida: c
   Obj: elimina el valor máximo y el mínimo de c *)

 function mayor (c:Tyconj): Tyele;
 (* Valor devuelto: mayor elemento del conjunto c
    Pre: c no es vacío *)
 var e : Tyele;
 begin
   e := fin;
   while not e in c do e := pred(e);
   mayor := e
 end;

 function menor (c: Tyconj): Tyele;
 (* Valor devuelto: menor elemento del conjunto c
    Pre: c no es vacío *)
 var e, min: Tyele;
 begin
   min := fin:
   begin
      for e := ini to fin do
          if (e in conj) and (e < min) then min := e;
      menor := min
 end;

begin
   if c <> [] then c := c - [mayor(c)] - [menor(c)]
end;
```

3 Cuestión 3

3.1

- Procedimiento.
- Entrada: 6 enteros h1, m1, s1 (primer tiempo) h2, m2, s2 (segundo tiempo).
- Salida: 3 enteros h3, m3, s3 (resultado de la suma).
- Precondición: $0 \leq m1, m2 \leq 59$ y $0 \leq s1, s2 \leq 59$ y $h1 + h2 \leq MAXINT$
- Cabecera: `procedure sumarTiempos (h1, m1, s1, h2, m2, s2: integer; var h3, m3, s3 : integer);`
- Llamada: dado `var p, q, r, t, a, b, c : integer;`:
  `sumartiempos (p, q+r, t, 1, 10, 59, a, b, c);`

3.2

- Función o procedimiento
- Entrada: inf, sup: caracteres extremos del rango
- Salida (si procedimiento): opción elegida, carácter
- Valor devuelto (si función): opción elegida, carácter
- Precondición: inf $\leq$ sup
- Cabecera: `function OpcionLeida (inf, sup: char): char`
  ó `procedure LeerOpcion (inf, sup: char; var opcion: char);`
- Llamada: Dado `var op: char`:
  `op := OpcionLeida('a', 'h');`
  o `LeerOpcion ('a', 'h', op)`

3.3

- Procedimiento
- Entrada:

  linea: cadena de caracteres (`string`)
  fichero en el que se escribe (`text`)
  ancho: anchura de la línea, entero
- Salida: fichero modificado
- Precondición: fichero preparado para escritura y a `ancho>=length(linea)`
- Cabecera

  ```
  procedure escribeAjustado
        (var t :  text; linea:  string; ancho:  integer);
  ```
- Llamada: dado `var t :  text` y `t := output)`

  ```
  escribeAjustado (t, 'Soy Ernesto', 15);
  ```

4 Cuestión 4

4.1

Registro:

```
type tpaciente = record
                     edad : integer;
                     peso, altura : real;
                     actividad : 0 .. 5;
                     ingesta : integer
                 end;
```

4.2

Matriz de registros, con un campo para el número y otro para el tachado:

```
const NLIN = ... ; NCOl = ... ;
type tcasilla = record
                   numero : integer;
                   tachado : boolean
                end;
     tcarton = array [1 .. NLIN, 1 .. NCOL] of tcasilla;
```

4.3

Fichero de resgistros. Supuesta la definición del apartado 1:

```
tficha = record
              ident : integer; (* identificador de paciente *)
              nombre : string;
              datos : tpaciente
         end;
tfichero = file of tficha;
```

5 Cuestión 5

```
function producto (a, b: integer): integer;
(* Obj: producto recursivo
   Pre: b >= 0
   Post : producto = a*b
*)
begin
    if b = 0 then producto := 0
    else if b=1 then producto := a
    else producto := a * producto(a, b-1)
end;
```

La función "termina" porque, si b es 0 ó 1, no hay más llamadas (casos básicos) y si no (b>0) la sucesión de llamadas se hace sobre una sucesión de enteros positivos decreciente (de 1 en 1), que forzosamente acabará en b=1, que es un caso básico.

6 Problema 1

Entrada: Vector de monedas en pesetas
Salida: Vector de monedas en euros
Objetivo: Obtiene el mínimo número de monedas de euros correspondiente a la cantidad que se obtiene de las monedas de pesetas.
Variables: CantidadP: entero para la cantidad de entrada
CantidadE: real, para la cantidad en euros
Subprogramas:

función ValorP (vector-monedas-peseta):
devuelve el valor correspondiente a un vector de número de monedas de pesetas, según el valor de cada una de ellas.

función real aEuros (entero cantidad-pesetas):
devuelve la cantidad-pesetas en euros, redondeando a 2 decimales

procedimiento DistribuyeE(entrada real cantidad-en-euros, salida vector-monedas-euro):
Distribuye la cantidad-en-euros en el mínimo número de monedas de euro

```
procedure PesAEuros (vp: TVectorPesetas; var ve : TVectorEuros);
(* posición de las funciones y el procedimiento que se definen
   más adelante *)
const CAMBIO=166.386;
var   CantidadP: integer;
      CantidadE: real;
begin
     CantidadP := Valor(vp);
     CantidadE := aEuros (CantidadP);
     DistribuyeE (CantidadE, ve)
end;
```

función ValorP
Entrada: vector de tipo `TVectorPesetas`
Valor devuelto: entero
Objetivo: devuelve el valor correspondiente a un vector de número de monedas de pesetas, según el valor de cada una de ellas.
Variables: provisional: entero, acumulador del valor a obtener
indice: de recorrido a lo largo de las monedas
ValorMoneda: vector con el valor de cada moneda
Subprogramas: procedimiento que asigna el valor de las monedas

```
function ValorP (m: TVectorPesetas): integer;
(* posición para el procedimiento AsignaValoresPesetas *)
var provisional: integer;
    indice: TIndicePesetas;
    ValorMoneda: TVectorPesetas; (* valor de cada moneda *)
begin
    AsignaValoresPesetas (ValorMoneda);
    provisional := 0;
    for indice := p1 to p500 do
        provisional := provisional + m[indice]*ValorMoneda[indice];
    ValorP := provisional
end;

procedure AsignaValoresPesetas (var ValorMoneda: TVectorPesetas);
var i: TIndicePesetas;
begin
    ValorMoneda[p1]:=1;
    ValorMoneda[p5]:=5;
    ValorMoneda[p10]:=10;
    ValorMoneda[p25]:=25;
    ValorMoneda[p50]:=50;
    ValorMoneda[p100]:=100;
    ValorMoneda[p200]:=200;
    ValorMoneda[p500]:=500;
end;
```

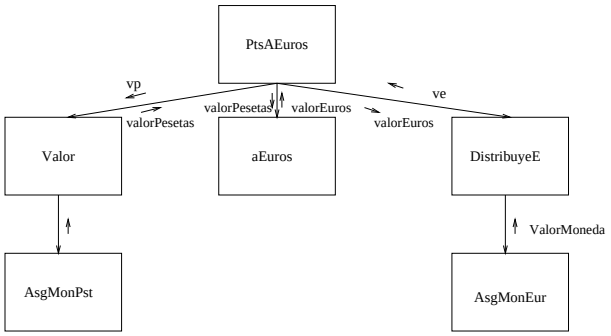
función aEuros
Entrada: cantidad en pesetas, entera
Valor devuelto: cantidad correspondiente en Euros, redondeada a dos decimales
Constantes: CAMBIO: 166'386 (ptas por euro)
Variables: centimos, entera

```
function aEuros (p: integer): real;
const CAMBIO=166.386;
var centimos : integer;
begin
    centimos := round( p/CAMBIO*100 );
    aEuros := centimos/100
end;
```

procedimiento DistribuyeE
Entrada: Cantidad en euros, real
Salida: vector de monedas, con la cantidad de entrada distribuida
en el mínimo número de monedas
Subprogramas: procedimiento que signa el valor a las monedas
Variables: ValorMoneda: vector con el valor de cada moneda, en céntimos
centimos: cantidad de entrada en céntimos
índice de recorrido del vector de monedas

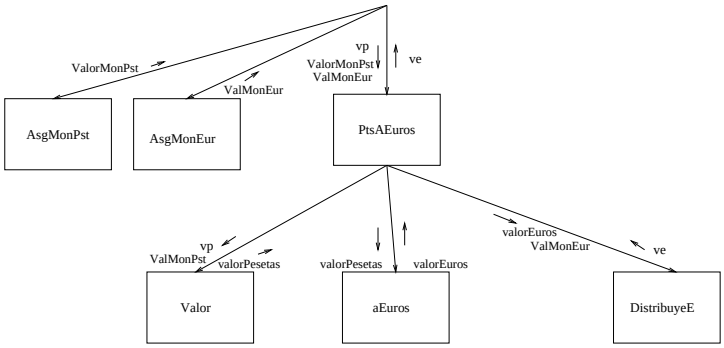
```
procedure DistribuyeE(e: integer; var ve: TVectorEuros);
(* posición para el procedimiento AsignaValoresCEuros *)
var   ValorMoneda: TVectorEuros;
      i: TIndiceEuros;
begin
    AsignaValoresCEuros (ValorMoneda);
    centimos := e * 100;
    for i := e2 downto c1 do
        begin
             ve[i] := e DIV ValorMoneda[i];
             centimos := centimos MOD ValorMoneda[i]
        end;
end;

procedure AsignaValoresCEuros (var ValorMoneda: TVectorEuros);
var i: TIndicePesetas;
```




```
begin
    ValorMoneda[c1]:=1;
    ValorMoneda[c2]:=2;
    ValorMoneda[c5]:=5;
    ValorMoneda[c10]:=10;
    ValorMoneda[c20]:=20;
    ValorMoneda[c50]:=50;
    ValorMoneda[e1]:=100;
    ValorMoneda[e2]:=200;
end;
```

Puede haber muchas otras alternativas. Una más eficiente si la transformación se va a usar varias veces iniciaría los valores de monedas una sola vez, de forma que la transformación requeriría dichos valores, según el siguiente diagrama:



7 Problema 2

7.1 Solución 1: por palabras

La idea es:

mientras no ff(entrada) **hacer**
    si línea no vacía **entonces**
        repetir
            leer palabra; reflejar palabra; escribir palabra;
            leer y copiar posibles blancos
        hasta fin de línea
    fin si
    leer y escribir fin de línea
fin mientras

Pero el problema es que, para leer una palabra completa, es necesario leer el carácter siguiente (blanco) o que se haya terminado la línea. Es decir, un procedimiento que leyera una palabra, dejaría leído el primer carácter del bloque de blancos. Por lo tanto, el procedimiento de copia de blancos partiría de haber leído ya el primer blanco y terminaría al leer el primer carácter de la siguiente palabra o al alcanzar el fin de línea. De la misma forma, el procedimiento de completar una palabra debería ser invocado cuando ya se ha leído el primer carácter de la misma.

Los procedimientos de obtención de palabra y de copia de blancos deben devolver, entonces, el carácter del siguiente bloque (grupo de blancos o palabra), caso de que lo haya y un indicador que determine si efectivamente lo hay.

Replanteando la solución con estos condicionantes:

mientras no ff(entrada) **hacer**
    (* línea *)
    si línea no vacía **entonces**
        leer carácter en palabra (se sabe que no empieza con b)
        repetir
            si hay más completar palabra; reflejar palabra; escribir palabra
            (* leído el primer b o en fin de línea *)
            copiar blancos si hay
            (* leído el primer carácter de la palabra o en f. de línea *)
        hasta que no haya más en la línea
    fin si
    leer y escribir fin de línea
fin mientras

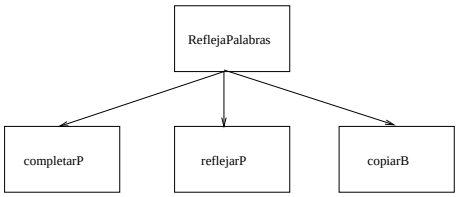
Se necesita una variable de tipo `string` para la palabra, otra de tipo `char` para el carácter que siempre se tiene leído de forma "adelantada" y además, se usará una variable lógica para determinar si efectivamente hay tal carácter leído o no lo hay (porque se ha alcanzado ya el fin de línea)

```
program ReflejaPalabras (f, g);
(*
   Entrada : texto: 'entrada.txt' (f)
   Salida  : texto en el que cada palabra está reflejada: 'salida.txt' (g)
   Pre : Las líneas no vacías no comienzan con blancos
*)
var f, g : text;
    palabra : string;
    c : char;
    hayMas : boolean;
Begin
    assign (f, 'entrada.txt'); reset (f);
    assign (g, 'salida.txt'); rewrite (g);
    while not eof (f) do
       begin (* línea *)
           if not eoln (f) then
              begin
                   read (f, c); (* no blanco *)
                   repeat
                        palabra := c;
                        if not eoln(f) then
                             completarP (f, palabra, hayMas);
                        reflejarP (palabra);
                        write (g, palabra);
                        if hayMas  then
                             copiarB (f, g, c, hayMas)
                   until not hayMas;
              end;
           readln (f); writeln (g)
       end;
    close (f); close (g)
End.
```

Módulos

módulo	**E**	**S**	**otros**	**obj**
completarP	- fichero - palabra de long. 1	- HayMas - palabra	avanza en f no está en eoln	- completa palabra - leído el primer b. del sgte bloque - hayMas indica si hay tal b
copiarB	- fich E - fich S	- c (primer c. sgte. si lo hay) - HayMas	avanza en f no está en eoln	- copia blancos - devuelve el primer c. del sgte bloque - hayMas indica si lo hay - escribe un b ya leído
reflejarP	- palabra	- palabra		refleja el parámetro

```
procedure completarP (var f: text; var pal: string; var hayMas: boolean);
(*
   Entrada f: texto
   Entrada-salida : pal:
              entra con un sólo carácter y sale con la palabra completa
   Salida : hayMas:  indica si hay blancos tras la palabra
   Obj : dada pal, compuesta de un carácter, completa la palabra
         que está en f, hasta un blanco
   Pre : f no está sobre el fin de línea
   Post : si tras a palabra hay un b. ya está leído
*)
var c : char;
begin
   c := pal[1];
   while not eoln (f) and c <> ' ' do
      begin
           read (f, c);
           if c <> ' ' then pal := pal + c
      end;
   if c=' ' then hayMas := TRUE
   else hayMas := FALSE
end;
```




```
procedure copiarB (var f, g : text; var c : char; var hayMas: boolean);
(*
   Entrada : fichero f, en determinada posición
   Salida : fichero g, en determinada posición
            c: primer carácter de la palabra siguiente, si la hay
            hayMas: indica si hay palabra siguiente
   Obj: copia blancos de f a g
        en c: el primer carácter no blanco a partir
              de la posición inicial de f
*)
begin
    write (g, ' '); c := ' ';
    while (not eoln(f)) and (c= ' ') do
       begin
          read (f, c);
          if c=' ' then write (g, ' ');
       end;
    if c<>' ' then hayMas := TRUE
    else hayMas := FALSE
end;

procedure reflejarP (var pal: string);
(* Entrada-Salida: pal (sale reflejada) *)
var i : integer;
    aux : char;
begin
    for i := 1 to lenght(pal) div 2 do
        begin
            aux := pal[i] ;
            pal[i] := pal[length(pal)+1-i] ;
            pal[length(pal)+1-i]i := aux ;
        end
end;
```

```
procedure reflejarP (var pal: string);
(* Entrada-Salida: pal (sale reflejada) *)
(* Recursivo *)
var subp : string;
begin
   if length(pal)>1 then
      begin
         subp := copy (pal,2, length(pal)-1);
         reflejarP (subp);
         pal := subp + pal[1]
      end
end;
```

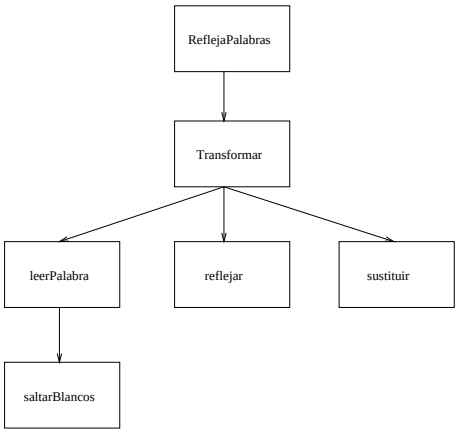
7.2 Solución 2: por líneas

La idea es:

mientras no ff(entrada) **hacer**
    leer línea de f
    transformar línea
    escribir línea en g
fin mientras

```
program ReflejaPalabras (f, g);
(*
   Entrada : texto: 'entrada.txt' (f)
   Salida  : texto en el que cada palabra está reflejada: 'salida.txt' (g)
   Pre : Las líneas no vacías no comienzan con blancos
*)
var f, g : text;
    linea : string;

Begin
    assign (f, 'entrada.txt'); reset (f);
    assign (g, 'salida.txt'); rewrite (g);
    while not eof (f) do
       begin (* línea *)
           readln (f, linea);
           transformarL (linea);
           writeln (g, linea);
```




```
        end;
End.
```

Subprogramas

módulo	**E**	**S**	**otros**	**obj**
transformarL	- linea	- linea		- refleja cada palabra
leerPalabra	- linea - inicial	- palabra - siguiente		- lee palabra de linea a partir de inicial - siguiente=posición de la sgte palabra
saltarBlancos	- linea - posicion	- posicion	sobre un b.	- salta blancos - obtiene posición del sgte no blanco o supera la longitud de la línea
reflejarP	- palabra	- palabra		refleja el parámetro
sustituir	- línea - inicial - palabra	- línea		- sustituye en línea el contenido a partir de incial por palabra

`reflejarP` está descrito en la solución anterior.

```
procedure transformarL (var linea: string);
(*
   Obj: refleja cada palabra de la línea.
        Una palabra es una sucesión de símbolos separados por blancos
   E-S : linea
*)
var palabra: string;
    inicial,  (* posición de comienzo de palabra *)
    siguiente : (* posición de comienzo de palabra sgte *)
                integer;
begin
    if length(linea)>0 then
       begin
          inicial:= 1;
          repeat
               leerPalabra (linea, inicial, palabra, siguiente);
               reflejarP (palabra);
               sustituir (linea, inicial, palabra);
               inicial := siguiente
          until inicial > length(linea)
       end
end;

procedure sustituir (var linea: string; inicial : integer; palabra: string);
(*
   Obj: coloca la palabra a partir de la posición inicial en la línea
        en sustitución de lo que haya en tales posiciones
   Pre: la palabra cabe en la línea a partir de la posición especificada
        (es decir: inicial+length(palabra)-1 <= length(linea))
   E: linea, inicial, palabra
   S: linea (modificada)
*)
var i : integer;
begin
   for i := 1 to length(palabra) do
       linea [ inicial+i-1 ] := palabra[i]
end;
```

```
procedure leerPalabra (linea: string; inicial: integer;
                       var palabra: string; var siguiente: integer);
(*
   Obj: lee la palabra que comienza en la posición 'inicial' de la línea
        obtiene en 'siguiente' la posición de comienzo
              de la siguiente palabra
        si no hay más palabras,
             'siguiente' tendrá un valor mayor que la longitud de línea
   Pre: inicial <= longitud (línea)
   E: linea, inicial
   S: palabra, siguiente
*)
var i : integer;
begin
    palabra := ''; i:= inicial;
    while (i < length(linea) and (linea[i]<>' ') do
          begin
              palabra := palabra + linea[i];
              i := i+1
          end;
    (* i >= length(linea) or i<length(linea) and linea[i]=' ' *)
    (* linea[i] no se ha comprobado *)
    if i = length(linea) then
       if linea[i]<>' ' then
          begin
             palabra := palabra + linea[i];
             i := i+1
          end;
    (* i señala el primer carácter posterior a la palabra
       posiblemente fuera de ella *)
    if i<= length(linea) then
          saltarBlancos (linea, i);
    siguiente := i
end;

procedure saltarBlancos (linea: string; var posición: integer);
(*
   Obj: dada un posición en la línea, avanza ésta hasta la primera
        posición en que no hay un blanco, o devuelve longitud(linea) +1
   Pre: linea[posicion] = blanco  i<length(linea)
*)
```

```
var i : integer;
begin
    i := posicion;
    while (i<length(linea)) and (linea[i]=' ') do i := i+1;
    if i = length(linea) then
          if linea[i]=' ' then i := i+1;
    posicion := i
end;
```

7.3 Solución 3: por líneas, de otra manera

```
program ReflejaPalabras (f, g);
(*
   Entrada : texto: 'entrada.txt' (f)
   Salida  : texto en el que cada palabra está reflejada: 'salida.txt' (g)
   Pre : Las líneas no vacías no comienzan con blancos
*)
var f, g : text;
    linea : string;

Begin
    assign (f, 'entrada.txt'); reset (f);
    assign (g, 'salida.txt'); rewrite (g);
    while not eof (f) do
       begin (* línea *)
           readln (f, linea);
           transferirL (linea, g);
           writeln (g, linea);
       end;
End.

procedure transferirL (linea: string, var g: text);
(*
   Obj: Copia la reflejada de cada palabra de la línea en g.
        Una palabra es una sucesión de símbolos separados por blancos.
        Copia los blancos según aparecen.
        Vacía la línea según va transfiriendo a g
   E : linea
   S : g
*)
```

```
var palabra: string;
    posb : integer; (* posición del primer blanco *)
begin
    if length(linea)>0 then
       begin
          repeat
               posb := pos(' ', linea);
               if posb = 0 then
                      palabra := copy (linea, 1, length(linea))
               else
                      palabra := copy (linea, 1, posb-1);
               reflejarP (palabra);
               write (g, palabra);
               delete (linea, 1, length(palabra));
               (* copiar blancos iniciales *)
               while (length(linea)>1) and (linea[1]=' ') do
                     begin
                            write (g, ' ');
                            delete (linea, 1, 1)
                     end;
               (* length(linea)=1 or >1 y linea[1] <> ' ' *)
               if linea[1] = ' ' then
                  begin
                        write (g, ' ');
                        delete (linea, 1, 1)
                  end
          until length(linea)=0;
       end;
end;
```

`reflejarP` está descrito en la solución 1.

7.4 Solución 4: por caracteres

El fichero se lee carácter a carácter. Cada vez que la lectura cambia de no blancos a blancos, se ha terminado una palabra. Cada vez que la lectura cambia de blancos a no blancos, comienza una palabra:

```
program ReflejaPalabras (f, g);
(* Entrada : texto: 'entrada.txt' (f)
   Salida  : texto en el que cada palabra está reflejada: 'salida.txt' (g)
```

```
   Pre : Las líneas no vacías no comienzan con blancos
*)
var f, g : text;
    palabra : string;
    c : char;
    EnPalabra : boolean;
Begin
    assign (f, 'entrada.txt'); reset (f);
    assign (g, 'salida.txt'); rewrite (g);
    while not eof (f) do
          begin (* línea *)
                EnPalabra := TRUE; palabra := ''; (* ver Pre *)
                while not eoln(f) do
                      begin
                          read (f, c);
                          (* tratar c: ver a continuación *)
                      end;
                readln(f); writeln(g)
          end;
    close (f); close(g)
End.
```

Refinamiento de `tratar c`

```
      if EnPalabra then
                if c<>' ' then palabra := palabra + c
                else (* palabra completada *)
                     begin
                           reflejarP (palabra);
                           write (g, palabra);
                           enPalabra := FALSE;
                           palabra := ''
                     end
      else (* fuera de palabra *)
                if c=' ' then write (g, c)
                else (* comienza palabra *)
                     begin
                           palabra := c;
                           enPalabra := TRUE
                     end
```

`reflejarP` está descrito en la solución 1.