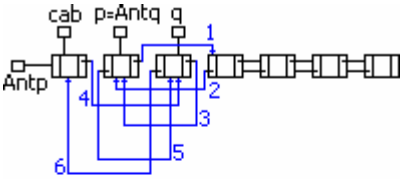
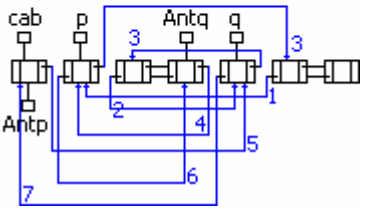
1. Dada una **lista doblemente enlazada** escribe un procedimiento que **modificando solamente los punteros** realice la ordenación del contenido de la lista **de mayor a menor** mediante la técnica conocida como **ordenación por selección**. (2,5 puntos)

Solución:

```
procedure Ordenar_por_seleccion_swap_punteros ( L : in out TLista ) is

    procedure Swap ( a, b : in out TPuntero ) is
       Aux : TPuntero := a;
    begin
       a := b;
       b := Aux;
    end;

    procedure Intercambiar ( p, q : in out TPuntero ) is
       Antp : TPuntero := p.Ant;
       Antq : TPuntero := q.Ant;
    begin
       if Antq=p then -- p y q son cajas consecutivas
          p.Sig := q.Sig; --1
          if q.Sig /= null then
             q.Sig.Ant := p; --2
          end if;
          q.Sig := p; --3

          if Antp /= null then
             Antp.Sig := q; --4
          end if;
          p.Ant := q; --5
          q.Ant := Antp; --6
       else
          if q.Sig /= null then
             q.Sig.Ant := p; --1
          end if;
          p.Sig.Ant := q; --2
          Swap ( p.Sig, q.Sig ); --3

          Antq.Sig := p; --4
          if Antp /= null then
             Antp.Sig := q; --5
          end if;
          p.Ant := Antq; --6
          q.Ant := Antp; --7
       end if;

       if p=L.cab then L.cab := q;
       end if;
       Swap ( p, q ); -- Restauro el recorrido
    end;

    p      : TPuntero := L.cab;
    q,
    pmayor : TPuntero := null;
    mayor  : Integer;
 begin
    if p=null or else p.Sig=null then return;
    end if;

    while p.Sig/=null loop
       mayor  := p.Dato;
       pmayor := p;
       q      := p.Sig;
       while q/=null loop
          if q.Dato > mayor then -- Swap de punteros
             mayor  := q.Dato;
             pmayor := q;
          end if;
          q := q.Sig;
       end loop;
       if pmayor /= p then
          Intercambiar ( p, pmayor );
       end if;
       p := p.Sig;
    end loop;
 end;
```





Gon Perez-Jimenez © 2004 Todos los derechos reservados

2. Escribe un procedimiento que recibe como parámetro de entrada un puntero P a un nodo de un árbol y, realizando un recorrido **recursivo**, devuelva una copia de todos los nodos del subárbol de P. (2,5 puntos)

Solución:

Siendo la estructura del ABB:

```
type TNodo;
type TPuntero is access TNodo;

type TNodo is record
        Dato         : TDato;
        Izq, Der     : TPuntero := null;
        Key          : Natural  := 0;
     end record;

type TArbol is record
        Root         : TPuntero := null;
        NumElementos : Natural  := 0;
     end record;
```

La rutina de copia será: (P = SubRoot)

```
procedure Copia ( SubRoot : in TPuntero; SubTree : out TArbol ) is

   Count : Natural := 0;

   function Copia ( Root: in TPuntero ) return TPuntero is
   begin
      if Root = null then return null;
      else
         Count := Count + 1;
         return new TNodo'(
                           Root.Dato,
                           Copia ( Root.Izq ),
                           Copia ( Root.Der ),
                           Root.Key
                           );
      end if;
   end;

begin
   SubTree := ( Copia ( SubRoot ) , Count );
end;
```

Gon Perez-Jimenez © 2004 Todos los derechos reservados

3. Dado un **grafo** representado mediante su matriz de adyacencia en el que los vértices se nombran mediante letras (A, B, C,...) y los enlaces tienen asociado un peso (Natural), escribe un procedimiento que muestre en pantalla **todos los posibles caminos** desde un determinado nodo (Origen) hasta un nodo Destino**, junto con el coste de cada camino**. (3 puntos)

Solución:

Las estructuras básicas de control del grafo serán:

```
type TNodo;
type TPuntero is access TNodo;
type TNodo is record
      Dato : Character;
      Sig  : TPuntero  := null;
     end record;

type TCamino is record
      Cab   : TPuntero := null;
      Coste : Natural  := 0;
     end record;

Infinito : constant Natural := Natural'Last;
type TGrafo is array ( Character range <>, Character range <> ) of Natural;
```

La estructura TCamino será usada para insertar en la lista simple al final cada nuevo nodo (character) que forme parte del camino. De ese modo podremos mostrar el camino al final respetando el orden en el que cada nodo fue insertado en el mismo.

Las rutinas básicas de control de la lista simple con cabecera (TCamino) son:

```
procedure Liberar is new Unchecked_Deallocation ( TNodo, TPuntero );

procedure Insertar_en_cabeza ( cab: in out TPuntero; Dato : Character ) is
begin
   cab := new TNodo'(Dato,cab);
end;

procedure Insertar_al_final ( cab: in out TPuntero; Dato : Character ) is
begin
   if cab=null then
      Insertar_en_cabeza ( cab, Dato );
   else
      Insertar_al_final ( cab.Sig, Dato );
   end if;
end;

function Existe ( cab : in TPuntero; Dato : Character ) return Boolean is
   p: TPuntero := cab;
begin
   while p/=null loop
      if p.Dato=Dato then return True;
      end if;
      p := p.Sig;
   end loop;
   return False;
end;

procedure Put ( cab : in TPuntero ) is
begin
   if cab=null then return;
   else
      Put ( cab.Dato );
      Put ( cab.Sig );
   end if;
end;

procedure Reset ( cab : in out TPuntero ) is
   p : TPuntero;
begin
   while cab/= null loop
      p := cab;
      cab := cab.Sig;
      p.Sig := null;
      Liberar (p);
   end loop;
end;
```

Gon Perez-Jimenez © 2004 Todos los derechos reservados

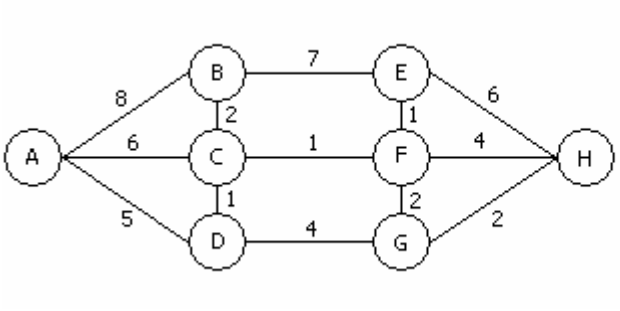
```
function Copiar ( Camino : in TCamino ) return TCamino is

    function Copiar ( cab : in TPuntero ) return TPuntero is
    begin
       if cab=null then return null;
       else
          return new TNodo'(cab.Dato, Copiar(cab.Sig) );
       end if;
    end;

 begin
    return ( Copiar(Camino.cab), Camino.Coste );
 end;

 procedure Añadir ( Camino     : in out TCamino;
                    Nodo       : in     Character;
                    Coste_nodo : in     Natural    ) is
 begin
    if Existe ( Camino.cab, Nodo ) then return;
    else
       Insertar_al_final ( Camino.cab, Nodo );
       Camino.Coste := Camino.Coste + Coste_nodo;
    end if;
 end;
```

Por último, la rutina que mostrará todas las combinaciones posibles para llegar desde el nodo Origen hasta el nodo Destino es:

```
   procedure Mostrar_Caminos ( Grafo   : in TGrafo;
                               Origen  : in Character;
                               Destino : in Character ) is

      procedure Avanzar ( Nodo       : in Character;
                          Coste_nodo : in Natural;
                          Camino     : in TCamino    ) is

         C  : TCamino := Copiar (Camino);
      begin
         Añadir ( C, Nodo, Coste_nodo );
         if Nodo = Destino then
            -- Muestro el camino seguido para alcanzar el nodo
            New_Line;
            Put ( C.cab );
            New_Line;
            Put_Line ( "Coste: " & Natural'Image ( C.Coste ) );
         else
            -- Aun no he llegado al destino. Marco los nodos de los vecinos
            for vecino in Grafo'Range(2) loop
               if vecino/=Origen and then
                  vecino/=Nodo   and then
                  Grafo(Nodo,vecino)/=Infinito and then
                  not Existe ( C.cab, vecino ) then
                  -- De forma recursiva sigo avanzando por este camino
                  Avanzar ( Vecino, Grafo(Nodo,Vecino), C );
               end if;
            end loop;
         end if;
         Reset ( C.cab );
      end;

      C  : TCamino;
   begin
      Avanzar ( Origen, 0, C );
   end;
```

Gon Perez-Jimenez © 2004 Todos los derechos reservados

Una llamada de ejemplo a esta rutina sería (Ejemplo de programa principal):



	A	B	C	D	E	F	G	H
A		8	6	5				
B	8		2		7			
C	6	2		1		1		
D	5		1				4	
E		7				1		6
F			1		1		2	4
G				4		2		2
H					6	4	2	

Matriz de adyacencia (Pesos)

```
Grafo : TGrafo ('A'..'H', 'A'..'H') := (
                                        ('B'=>8, 'C'=>6, 'D'=>5, others => Infinito),
                                        ('A'=>8, 'C'=>2, 'E'=>7, others => Infinito),
                                        ('A'=>6, 'B'=>2, 'D'=>1, 'F'=>1 , others => Infinito),
                                        ('A'=>5, 'C'=>1, 'G'=>4, others => Infinito),
                                        ('B'=>7, 'F'=>1, 'H'=>6 , others => Infinito),
                                        ('C'=>1, 'E'=>1, 'G'=>2, 'H'=>4, others => Infinito),
                                        ('D'=>4, 'F'=>2, 'H'=>2, others => Infinito),
                                        ('E'=>6, 'F'=>4, 'G'=>2, others => Infinito)
                                      );

begin
 Mostrar_Caminos ( Grafo, 'A', 'B' );
end;
```

Siendo el resultado en pantalla: Comprobando que la rutina funciona correctamente y es rápida

```
AB
Coste:  8

ACB
Coste:  8

ACDGFEB
Coste:  21

ACDGFHEB
Coste:  30

ACDGHEB
Coste:  26

ACDGHFEB
Coste:  25

ACFEB
Coste:  15

ACFGHEB
Coste:  24

ACFHEB
Coste:  24

ADCB
Coste:  8

ADCFEB
Coste:  15

ADCFGHEB
Coste:  24

ADCFHEB
Coste:  24

ADGFCB
Coste:  14

ADGFEB
Coste:  19

ADGFHEB
Coste:  28

ADGHEB
Coste:  24

ADGHEFCB
Coste:  21

ADGHFCB
Coste:  18

ADGHFEB
Coste:  23
```

Gon Perez-Jimenez © 2004 Todos los derechos reservados