Juan Menduiña

Resumen

0. CONSTANTES Y TIPOS:

```
program resumen;


{########## CONSTANTES ##########}


const

  {Se definen constantes de salida, de objetivos, de tamaño de vectores, etc.}


{########## TIPOS ##########}


type

  t_str20=string[20];
  t_rango_num=1..tam;
  t_rango_str1='a'..'z';
  t_rango_str2=(juan,ignacio);
  t_vector=array[t_rango_num] of integer;
  t_registro=record
    ele1: integer;
    ele2: string;
    ..
  end;
  t_lista=^t_nodo;
  t_nodo=record
    ele: {integer, string, record, array, etc.};
    sig: t_lista;
  end;
```

1. MÓDULOS REGISTROS:

```
{########## 1. REGISTROS ##########}


{LEER 1 (while) - REGISTROS}

procedure leer1(var registro: t_registro);
begin
  readln(registro.ele1);
  if (registro.ele1<>ele1_salida) then
    readln(registro.ele2);
end;

{LEER 2 (repeat-until) - REGISTROS}

procedure leer2(var registro: t_registro);
begin
  readln(registro.ele1);
  readln(registro.ele2);
end;

{IMPRIMIR - REGISTROS}

procedure imprimir(registro: t_registro);
begin
  writeln(registro.ele1);
  writeln(registro.ele2);
end;

{IGUALES - REGISTROS}

function iguales(registro, registro: t_registro): boolean:
begin
  iguales:=((registro.ele1=registro.ele1) and (registro.ele2=registro.ele2));
end;

{CORTE DE CONTROL 1 (while) - REGISTROS}

procedure corte_control1(registro: t_registro; valor: integer);
var
  cant_actual, total: integer;
  nomb_actual: string;
begin
  total:=0;
  leer(registro);
  while (registro.ele1<>ele1_salida) do
  begin
    cant_actual:=0;
    nomb_actual:=registro.ele2;
    while ((registro.ele1<>ele1_salida) and (registro.ele2=nomb_actual)) do
    begin
      cant_actual:=cant_actual+1;
      leer(registro);
    end;
    total:=total+cant_actual;
    writeln(cant_actual);
  end;
  writeln(total);
end;

{CORTE DE CONTROL 2 (repeat-until) - REGISTROS}

procedure corte_control2(registro: t_registro; valor: integer);
var
```

```
  cant_actual, total: integer;
  nomb_actual: string;
begin
  total:=0;
  repeat
    leer(registro);
    cant_actual:=0;
    nomb_actual:=registro.ele2;
    while (registro.ele2=nomb_actual) do
    begin
      cant_actual:=cant_actual+1;
      leer(registro);
    end;
    total:=total+cant_actual;
    writeln(cant_actual);
  until (registro.ele1=ele1_salida);
  writeln(total);
end;

{CORTE DE CONTROL 3 (for) - REGISTROS}

procedure corte_control3(registro: t_registro; var vector: t_vector);
var
  i, j: integer;
  nomb_actual: string;
begin
  j:=1
  for i:= 1 to max_reg do
  begin
    leer(registro);
    nomb_actual:=registro.ele2;
    if (registro.ele2<>nomb_actual) then
      j:=j+1;
    vector[j]:=vector[j]+registro.ele1;
  end;
end;
```

2. MÓDULOS VECTORES:

```
{########## 2. VECTORES ##########}


{CARGAR 1 (for) - VECTORES}

procedure cargar1(var vector: t_vector; dimL: integer);
var
  i: integer;
begin
  for i:= 1 to dimL do
    readln(vector[i]);
end;

{CARGAR 2 (while) - VECTORES}

procedure cargar2(var vector: t_vector; var dimL: integer);
var
  num: integer;
begin
  readln(num);
  while ((dimL<dimF) and (num<>vector_salida)) do
  begin
    dimL:=dimL+1;
    vector[dimL]:=num;
    readln(num);
  end;
end;

{CARGAR 3 (repeat-until) - VECTORES}

procedure cargar3(var vector: t_vector; var dimL: integer);
var
  num: integer;
begin
  repeat
    readln(num);
    dimL:=dimL+1;
    vector[dimL]:=num;
  until ((dimL=dimF) of (num=vector_salida));
end;

{IMPRIMIR - VECTORES}

procedure imprimir(vector: t_vector);
var
  i: integer;
begin
  for i:= 1 to dimL do
     writeln(v[i]);
end;

{MÁXIMO - VECTORES}

function maximo(vector: t_vector; dimL: integer): integer;
var
  i, val_max: integer;
begin
  val_max:=low(integer);
  for i:= 1 to dimL do
    if (vector[i]>val_max) then
      max:=vector[i];
  maximo:=val_max;
end;
```

```
function maximo(vector: t_vector; dimL: integer): integer;
var
  i, val_max, pos_max: integer;
begin
  val_max:=low(integer);
  for i:= 1 to dimL do
    if (vector[i]>val_max) then
    begin
      val_max:=vector[i];
      pos_max:=i;
    end;
  maximo:=pos_max;
end;

procedure maximo(vector: t_vector; dimL: integer; var val_max, pos_max: integer);
var
  i: integer;
begin
  for i:= 1 to dimL do
  begin
    if (vector[i]>val_max) then
    begin
      val_max:=vector[i];
      pos_max:=i;
    end;
  end;
end;

{VERIFICAR 1 (existe) - VECTORES}

function verificar1(vector: t_vector; dimL, valor: integer): boolean;
var
  pos: integer;
begin
  while ((pos<=dimL) and (vector[pos]<>valor)) do
    pos:=pos+1;
  verificar1:=(pos<=dimL);
end;

{VERIFICAR 2 (cuántos existen) - VECTORES}

function verificar2(vector: t_vector; dimL, valor: integer): integer;
var
  i, cant: integer;
begin
  cant:=0;
  for i:= 1 to dimL do
    if (vector[i]=valor) then
      cant:=cant+1;
  verificar2:=cant;
end;

{RECORRER TOTAL (cuántos existen) - VECTORES}

function recorrer_total(vector: t_vector; dimL, valor: integer): integer;
var
  i, cant: integer;
begin
  cant:=0;
  for i:= 1 to dimL do
    if (vector[i]=valor) then
      cant:=cant+1;
  recorrer_total:=cant;
end;
```

```
{RECORRER PARCIAL 1 (vector desordenado, se sabe que existe) - VECTORES}

function recorrer_parcial1(vector: t_vector; valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while (vector[pos]<>valor) do
    pos:=pos+1;
  recorrer_parcial1:=pos;
end;

function recorrer_parcial1(vector: t_vector; valor: integer): integer;
var
  pos: integer;
  ok: boolean;
begin
  pos:=1; ok:=false;
  while (ok=false) do
    if (vector[pos]=valor) then
      ok:=true
    else
      pos:=pos+1;
  recorrer_parcial1:=pos;
end;

{RECORRER PARCIAL 2 (vector desordenado, no se sabe que existe) - VECTORES}

function recorrer_parcial2(vector: t_vector; dimL, valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while ((pos<=dimL) and (vector[pos]<>valor)) do
    pos:=pos+1;
  if (pos<=dimL) then
    recorrer_parcial2:=pos;
  else
    recorrer_parcial2:=-1;
end;

function recorrer_parcial2(vector: t_vector; dimL, valor: integer): integer;
var
  pos: integer;
  ok: boolean;
begin
  pos:=1; ok:=false;
  while ((pos<=dimL) and (ok=false)) do
    if (vector[pos]=valor) then
      ok:=true
    else
      pos:=pos+1;
  if (ok=true) then
    recorrer_parcial2:=pos
  else
    recorrer_parcial2:=-1;
end;

{RECORRER PARCIAL 3 (vector ordenado, se sabe que existe) - VECTORES}

function recorrer_parcial3(vector: t_vector; valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while (vector[pos]<valor) do
```

```
    pos:=pos+1;
  if (vector[pos]=valor) then
    recorrer_parcial3:=pos;
end;

{RECORRER PARCIAL 4 (vector ordenado, no se sabe que existe) - VECTORES}

function recorrer_parcial4(vector: t_vector; dimL, valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while ((pos<=dimL) and (vector[pos]<valor)) do
    pos:=pos+1;
  if ((pos<=dimL) and (vector[pos]=valor)) then
    recorrer_parcial4:=pos;
  else
    recorrer_parcial4:=-1;
end;

{AGREGAR - VECTORES}

procedure agregar(var vector: t_vector; var dimL: integer; var ok: boolean; num: integer);
begin
  if (dimL<dimF) then
  begin
    ok:=true;
    dimL:=dimL+1;
    vector[dimL]:=num;
  end;
end;

{INSERTAR - VECTORES}

procedure insertar(var vector: t_vector; var dimL: integer; var ok: boolean; num, pos:
integer);
var
  i: integer;
begin
  if ((dimL<dimF) and (pos>=1 and pos<=dimL)) then
  begin
    for i:= dimL dowto pos do
      vector[i+1]:=vector[i];
    ok:=true;
    vector[pos]:=num;
    dimL:=dimL+1;
  end;
end;

{ELIMINAR - VECTORES}

procedure eliminar(var vector: t_vector; var dimL: integer; var ok: boolean; pos: integer);
var
  i: integer;
begin
  if (pos>=1 and pos<=dimL) then
  begin
    for i:= pos to (dimL-1) do
      vector[i]:=vector[i+1];
    ok:=true;
    dimL:=dimL-1;
  end;
end;

{BUSCAR 1 (vector desordenado) - VECTORES}
```

```
function buscar1(vector: t_vector; dimL, valor: integer): boolean;
var
  pos: integer;
begin
  pos:=1;
  while ((pos<=dimL) and (vector[pos]<>valor)) do
    pos:=pos+1;
  buscar1:=(pos<=dim);
end;

function buscar1(vector: t_vector; dimL, valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while ((pos<=dimL) and (vector[pos]<>valor)) do
    pos:=pos+1;
  if (pos<=dimL) then
    buscar1:=pos
  else
    buscar1:=-1;
end;

{BUSCAR 2 (vector ordenado, no dicotómico) - VECTORES}

function buscar2(vector: t_vector; dimL, valor: integer): boolean;
var
  pos: integer;
begin
  pos:=1;
  while ((pos<=dimL) and (vector[pos]<valor)) do
    pos:=pos+1;
  buscar1:=((pos<=dim) and (vector[pos]=valor));
end;

function buscar2(vector: t_vector; dimL, valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while ((pos<=dimL) and (vector[pos]<valor)) do
    pos:=pos+1;
  if ((pos<=dimL) and (vector[pos]=valor)) then
    buscar2:=pos
  else
    buscar2:=-1;
end;

{BUSCAR 3 (vector ordenado, dicotómico) - VECTORES}

function buscar3(vector: t_vector; dimL, valor: integer): boolean;
var
  pri, ult, medio: integer;
begin
  pri:=1; ult:=dimL; medio:=(pri+ult) div 2;
  while ((pri<=ult) and (vector[medio]<>valor)) do
  begin
    if (vector[medio]>valor) then
      ult:=medio-1
    else
      pri:=medio+1;
    medio:=(pri+ult) div 2;
  end;
  buscar3:=(pri<=ult);
end;
```

```
function buscar3(vector: t_vector; dimL, valor: integer): integer;
var
  pri, ult, medio: integer;
begin
  pri:=1; ult:=dimL; medio:=(pri+ult) div 2;
  while ((pri<=ult) and (vector[medio]<>valor)) do
  begin
    if (vector[medio]>valor) then
      ult:=medio-1
    else
      pri:=medio+1;
    medio:=(pri+ult) div 2;
  end;
  if (pri<=ult) then
    buscar3:=medio
  else
    buscar3:=-1;
end;

{CONTADORES - VECTORES}

procedure inicializar(var vector_cont: t_vector_cont);
var
  i: integer;
begin
  for i:= min_cont to max_cont do
    vector_cont[i]:=0;
end;

procedure descomponer(var vector_cont: t_vector_cont; valor: integer);
begin
  while (valor<>0) do
  begin
    vector_cont[valor mod 10]:=vector_cont[valor mod 10]+1;
    valor:=valor div 10;
  end;
end;

procedure informar(vector_cont: t_vector_cont);
var
  i: integer;
begin
  for i:= min_cont to max_cont do
    writeln(vector_cont[i]);
end;

function pares_impares(vector_cont: t_vector_cont): boolean;
var
  i, pares, impares: int8;
begin
  pares:=0; impares:=0;
  for i:= min_cont to max_cont do
    if (vector_cont[i]<>0) then
      if (i mod 2=0) then
        pares:=pares+vector_cont[i]
      else
        impares:=impares+vector_cont[i];
  pares_impares:=(pares>impares);
end;

procedure maximo(vector_cont: t_vector_cont; var val_max, pos_max: integer);
var
  i: integer;
begin
  for i:= min_cont to max_cont do
  begin
```

```
    if (vector_cont[i]>val_max) then
    begin
      val_max:=vector_cont[i];
      pos_max:=i;
    end;
  end;
end;

{ORDENAR - VECTORES}

procedure ordenar(var vector: t_vector; dimL: integer);
var
  i, j, k, item: integer;
begin
  for i:= 1 to (dimL-1) do
  begin
    k:=i;
    for j:= (i+1) to dimL do
      if (vector[j]<vector[k]) then
        k:=j;
    item:=vector[k];
    vector[k]:=vector[i];
    vector[i]:=item;
  end;
end;
```

3. MÓDULOS LISTAS:

```
{########## 3. LISTAS ##########}


{CREAR - LISTA}

procedure crear(lista: t_lista);
begin
  lista:=nil;
end;

{RECORRER - LISTA}

procedure recorrer(lista: t_lista);
begin
  while (lista<>nil) do
  begin
    write(lista^.ele.ele1);
    lista:=lista^.sig;
  end;
end;

{AGREGAR ADELANTE - LISTA}

procedure agregar_adelante(var lista: t_lista; registro: t_registro);
var
  nuevo: t_lista;
begin
  new(nuevo);
  nuevo^.ele:=registro;
  nuevo^.sig:=lista;
  lista:=nuevo;
end;

procedure agregar_adelante(var lista: t_lista; registro: t_registro);
var
  nuevo: t_lista;
begin
  new(nuevo);
  nuevo^.ele:=registro;
  nuevo^.sig:=nil;
  if (lista=nil) then
    lista:=nuevo
  else
  begin
    nuevo^.sig:=lista;
    lista:=nuevo;
  end;
end;

{AGREGAR ATRÁS - LISTA}

procedure agregar_atras(var lista, ultimo: t_lista; registro: t_registro);
var
  nuevo: t_lista;
begin
  new(nuevo);
  nuevo^.ele:=registro;
  nuevo^.sig:=nil;
  if (lista=nil) then
    lista:=nuevo
  else
    ultimo^.sig:=nuevo;
  ultimo:=nuevo;
```

```
end;

{AGREGAR ORDENADO - LISTA}

procedure agregar_ordenado(var lista: t_lista; registro: t_registro);
var
  anterior, actual, nuevo: t_lista;
begin
  new(nuevo);
  nuevo^.ele:=registro;
  anterior:=lista; actual:=lista;
  while ((actual<>nil) and (actual^.ele.ele1<nuevo^.ele.ele1)) do
  begin
    anterior:=actual;
    actual:=actual^.sig;
  end;
  if (actual=lista) then
    lista:=nuevo
  else
    anterior^.sig:=nuevo;
  nuevo^.sig:=actual;
end;

procedure agregar_ordenado(var lista: t_lista; registro: t_registro);
var
  anterior, actual, nuevo: t_lista;
begin
  new(nuevo);
  nuevo^.ele:=registro;
  nuevo^.sig:=nil;
  anterior:=lista; actual:=lista;
  if (lista=nil) then
    lista:=nuevo
  else
  begin
    while ((actual<>nil) and (actual^.ele.ele1<nuevo^.ele.ele1)) do
    begin
        anterior:=actual;
        actual:=actual^.sig;
    end;
  end;
  if (actual=lista) then
  begin
    lista:=nuevo;
    nuevo^.sig:=actual;
  end
  else
  begin
    anterior^.sig:=nuevo;
    nuevo^.sig:=actual;
  end;
end;

{AGREGAR FUSIÓN - LISTA}

procedure agregar_fusion(avar lista, ultimo: t_lista; registro: t_registro; select:
t_select);
var
  anterior, actual, nuevo: t_lista;
begin

  new(nuevo);
  nuevo^.ele:=registro;
  nuevo^.sig:=nil;

  if (select=3) then
```

```
  begin
    actual:=lista;
    anterior:=lista;
  end;

  if (lista=nil) then
  begin
    lista:=nuevo
    if (select=2) then
      ultimo:=nuevo;
  end

  else
  begin

    if (select=1) then
    begin
      nuevo^.sig:=lista;
      lista:=nuevo;
    end;

    if (select=2) then
    begin
      ultimo^.sig:=nuevo;
      ultimo:=nuevo;
    end;

    if (select=3) then
    begin
      while ((actual<>nil) and (actual^.ele.ele1<nuevo^ele.ele1)) do
      begin
        anterior:=actual;
        actual:=actual^.sig;
      end;
    end;

  end;

  if (select=3) then
  begin
    if (actual=lista) then
      lista:=nuevo
    else
      anterior^.sig:=nuevo;
    nuevo^.sig:=actual;
  end;

end;

{LEER REGISTRO 1 (while) - LISTA}

procedure leer_registro1(var registro: t_registro);
begin
  write('Ingresar valor ele1: '); readln(registro.ele1);
  if (registro.ele1<>ele1_salida) then
  begin
    write('Ingresar valor ele2: '); readln(registro.ele2);
  end;
end;

{LEER REGISTRO 2 (repeat-until) - LISTA}

procedure leer_registro2(var registro: t_registro);
begin
  write('Ingresar valor ele1: '); readln(registro.ele1);
  write('Ingresar valor ele2: '); readln(registro.ele2);
```

```
end;

{CARGAR 1 (while) - LISTA}

procedure cargar1(var lista: t_lista);
var
  registro: t_registro;
  ultimo: t_lista;
begin
  leer_registro1(registro);
  while (registro.ele1<>ele1_salida) do
  begin
    agregar_adelante(lista,registro);
    {O}
    agregar_atras(lista,ultimo,registro);
    {O}
    agregar_ordenado(lista,registro);
    leer_registro1(registro);
  end;
end;

{CARGAR 2 (repeat-until) - LISTA}

procedure cargar2(var lista: t_lista);
var
  registro: t_registro;
  ultimo: t_lista;
begin
  repeat
    leer_registro2(registro);
    agregar_adelante(lista,registro);
    {o}
    agregar_atras(lista,ultimo,registro);
    {o}
    agregar_ordenado(lista,registro);
  until (registro.ele1=ele1_salida);
end;

{BUSCAR 1 (lista desordenada, boolean) - LISTA}

function buscar1(lista: t_lista; valor: integer): boolean;
begin
  while ((lista<>nil) and (lista^.ele.ele1<>valor)) do
    lista:=lista^.sig;
  buscar1:=(lista<>nil);
end;

function buscar1(lista: t_lista; valor: integer): boolean;
var
  ok: boolean;
begin
  ok:=false;
  while ((lista<>nil) and (ok=false)) do
    if (lista^.ele.ele1=valor) then
      ok:=true
    else
      lista:=lista^.sig;
  buscar1:=ok;
end;

{BUSCAR 2 (lista desordenada, nodo) - LISTA}

procedure buscar2(lista: t_lista; valor: integer; var pos: t_lista);
begin
  while ((lista<>nil) and (lista^.ele.ele1<>valor)) do
    lista:=lista^.sig;
```

```
  if (lista<>nil) then
    pos:=lista;
end;

{BUSCAR 3 (lista ordenada, boolean) - LISTA}

function buscar3(lista: t_lista; valor: integer): boolean;
begin
  while ((lista<>nil) and (lista^.ele.ele1<valor)) do
    lista:=lista^.sig;
  buscar3:=((lista<>nil) and (lista^.ele.ele1=valor));
end;

{BUSCAR 4 (lista ordenada, nodo) - LISTA}

procedure buscar4(lista: t_lista; valor: integer; var pos: t_puntero);
begin
  while ((lista<>nil) and (lista^.ele.ele1<valor)) do
    lista:=lista^.sig;
  if ((lista<>nil) and (lista^.ele.ele1=valor)) then
    pos:=lista;
end;

{ELIMINAR 1 (lista desordenada, sin repetición) - LISTA}

procedure eliminar1(var lista: t_lista; var elimino: boolean; valor: integer);
var
  anterior, actual: t_lista;
begin
  actual:=lista;
  while ((actual<>nil) and (actual^.ele.ele1<>valor)) do
  begin
    anterior:=actual;
    actual:=actual^.sig;
  end;
  if (actual<>nil) then
  begin
    if (actual=lista) then
      lista:=lista^.sig
    else
      anterior^.sig:=actual^.sig;
    dispose(actual);
    elimino:=true;
  end;
end;

{ELIMINAR 2 (lista desordenada, con repetición) - LISTA}

procedure eliminar2(var lista: t_lista; var elimino: boolean; valor: integer);
var
  anterior, actual: t_lista;
begin
  anterior:=lista; actual:=lista;
  while (actual<>nil) do
  begin
    if (actual^.ele.ele1<>valor) then
    begin
      anterior:=actual;
      actual:=actual^.sig;
    end
    else
    begin
      if (actual=lista) then
        lista:=lista^.sig
      else
        anterior^.sig:=actual^.sig;
```

```
      dispose(actual);
      actual:=anterior;
      elimino:=true;
    end;
  end;
end;

{ELIMINAR 3 (lista ordenada, sin repetición) - LISTA}

procedure eliminar3(var lista: t_lista; var elimino: boolean; valor: integer);
var
  anterior, actual: t_lista;
begin
  actual:=lista;
  while ((actual<>nil) and (actual^.ele.ele1<valor)) do
  begin
    anterior:=actual;
    actual:=actual^.sig;
  end;
  if ((actual<>nil) and (actual^.ele.ele1=valor)) then
  begin
    if (actual=lista) then
      lista:=lista^.sig
    else
      anterior^.sig:=actual^.sig;
    dispose(actual);
    elimino:=true;
  end;
end;

{ELIMINAR 4 (lista ordenada, con repetición) - LISTA}

procedure eliminar4(var lista: t_lista; var elimino: boolean; valor: integer);
var
  anterior, actual: t_lista;
begin
  anterior:=lista; actual:=lista;
  while ((actual<>nil) and (actual^.ele.ele1<=valor)) do
  begin
    if (actual^.ele.ele1<valor) then
    begin
      anterior:=actual;
      actual:=actual^.sig;
    end
    else
    begin
      if (actual=lista) then
        lista:=lista^.sig
      else
        anterior^.sig:=actual^.sig;
      dispose(actual);
      actual:=anterior;
      elimino:=true;
    end;
  end;
end;
```