*Juan Menduiña*

# Resumen

## 0. CONSTANTES Y TIPOS:

```
program resumen;


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


const

  const1=1;
  const2=1.0;
  const3='a';


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


type

  t_str10=string[10];
  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: real;
    ele3: 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
  begin
    readln(registro.ele2);
    readln(registro.ele3);
    ...
  end;
end;

{LEER 2 (repeat-until) - REGISTROS}

procedure leer2(var registro: t_registro);
begin
  readln(registro.ele1);
  readln(registro.ele2);
  readln(registro.ele3);
  ...
end;

{IMPRIMIR - REGISTROS}

procedure imprimir(registro: t_registro);
begin
  writeln(registro.ele1);
  writeln(registro.ele2);
  writeln(registro.ele3);
  ...
end;

{IGUALES - REGISTROS}

function iguales(registro, registro: t_registro): boolean:
begin
  iguales:=((registro.ele1=registro.ele1) and (registro.ele2=registro.ele2) and
(registro.ele3=registro.ele3) and ...);
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.ele3;
    while ((registro.ele1<>ele1_salida) and (registro.ele3=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.ele3;
    while (registro.ele3=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.ele3;
    if (registro.ele3<>nomb_actual) then
      j:=j+1;
    vector[j]:=vector[j]+registro.ele1;
  end;
end;
```

# 2. MÓDULOS VECTORES:

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


{INICIALIZAR - VECTORES}

procedure inicializar(var vector: t_vector);
var
  i: integer;
begin
  for i:= 1 to dimF do
    vector[i]:=0;
end;

{RECORRER - VECTORES}

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

{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<>num_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) or (num=num_salida));
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 downto pos do
      vector[i+1]:=vector[i];
    ok:=true;
    vector[pos]:=num;
    dimL:=dimL+1;
  end;
end;

{BUSCAR 1 (vector desordenado, no se sabe si existe) - 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<=dimL);
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 desordenado, se sabe que existe) - VECTORES}

function buscar2(vector: t_vector; valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while (vector[pos]<>valor) do
    pos:=pos+1;
  buscar2:=pos;
end;

{BUSCAR 3 (vector ordenado, no dicotómico, no se sabe si existe) - VECTORES}
```

```
function buscar3(vector: t_vector; dimL, valor: integer): boolean;
var
  pos: integer;
begin
  pos:=1;
  while ((pos<=dimL) and (vector[pos]<valor)) do
    pos:=pos+1;
  buscar3:=((pos<=dimL) and (vector[pos]=valor));
end;

function buscar3(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
    buscar3:=pos
  else
    buscar3:=-1;
end;

{BUSCAR 4 (vector ordenado, no dicotómico, se sabe que existe) - VECTORES}

function buscar4(vector: t_vector; valor: integer): integer;
var
  pos: integer;
begin
  pos:=1;
  while (vector[pos]<valor) do
    pos:=pos+1;
  buscar4:=pos;
end;

{BUSCAR 5 (vector ordenado, dicotómico, no se sabe si existe) - VECTORES}

function buscar5(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;
  buscar5:=(pri<=ult);
end;

function buscar5(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
    buscar5:=medio
  else
    buscar5:=-1;
end;

{BUSCAR 6 (vector ordenado, dicotómico, se sabe que existe) - VECTORES}

function buscar6(vector: t_vector; dimL, valor: integer): integer;
var
  pri, ult, medio: integer;
begin
  pri:=1; ult:=dimL; medio:=(pri+ult) div 2;
  while (vector[medio]<>valor) do
  begin
    if (vector[medio]>valor) then
      ult:=medio-1
    else
      pri:=medio+1;
    medio:=(pri+ult) div 2;
  end;
  buscar6:=medio;
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;

{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;

{MÁXIMO/MÍNIMO - VECTORES}

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

```
      val_max:=vector[i];          // val_min:=vector[i];
  maximo:=val_max;                 // minimo:==val_min;
end;

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

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

procedure maximo_minimo(vector: t_vector; dimL: integer; var val_max, pos_max, val_min,
pos_min: 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;
    if (vector[i]<val_min) then
    begin
      val_min:=vector[i];
      pos_min:=i;
    end;
  end;
end;
```

# 3. MÓDULOS LISTAS:

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


{INICIALIZAR - LISTAS}

procedure inicializar(var lista: t_lista);
begin
  lista:=nil;
end;

{RECORRER - LISTAS}

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

{AGREGAR ADELANTE - LISTAS}

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 - LISTAS}

procedure agregar_atras1(var lista, ult: 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
    ult^.sig:=nuevo;
  ult:=nuevo;
```

```
end;

procedure agregar_atras2(var lista: t_lista; registro: t_registro);
var
  nuevo, ult: t_lista;
begin
  new(nuevo);
  nuevo^.num:=registro;
  nuevo^.sig:=nil;
  if (lista=nil) then
    lista:=nuevo
  else
  begin
    ult:=lista;
    while (ult^.sig<>nil) do
      ult:=ult^.sig;
    ult^.sig:=nuevo;
  end;
end;

{AGREGAR ORDENADO - LISTAS}

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

```
    anterior^.sig:=nuevo;
    nuevo^.sig:=actual;
  end;
end;

{AGREGAR FUSIÓN - LISTAS}

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

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

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

  else
  begin

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

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

    if (select=3) then
    begin
      actual:=lista;
      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;

{CARGAR 1 (while) - LISTAS}

procedure cargar1(var lista: t_lista);
var
  registro: t_registro;
```

```
  ultimo: t_lista;
begin
  leer1(registro);
  while (registro.ele1<>ele1_salida) do
  begin
    agregar_adelante(lista,registro);     {Opción 1}
    agregar_atras(lista,ultimo,registro); {Opción 2}
    agregar_ordenado(lista,registro);     {Opción 3}
    leer1(registro);
  end;
end;

{CARGAR 2 (repeat-until) - LISTAS}

procedure cargar2(var lista: t_lista);
var
  registro: t_registro;
  ultimo: t_lista;
begin
  repeat
    leer2(registro);
    agregar_adelante(lista,registro);     {Opción 1}
    agregar_atras(lista,ultimo,registro); {Opción 2}
    agregar_ordenado(lista,registro);     {Opción 3}
  until (registro.ele1=ele1_salida);
end;

{BUSCAR 1 (lista desordenada, no se sabe si existe) - LISTAS}

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

{BUSCAR 2 (lista desordenada, se sabe que existe) - LISTAS}

procedure buscar3(lista: t_lista; valor: integer; var nodo: t_lista);
begin
  while (lista^.ele.ele1<>valor) do
    lista:=lista^.sig;
  nodo:=lista;
end;

{BUSCAR 3 (lista ordenada, no se sabe si existe) - LISTAS}

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;

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

```
    buscar3:=lista
  else
    buscar3:=nil;
end;

{BUSCAR 4 (lista ordenada, se sabe que existe) - LISTAS}

procedure buscar4(lista: t_lista; valor: integer): t_puntero;
begin
  while (lista^.ele.ele1<valor) do
    lista:=lista^.sig;
  buscar4:=lista;
end;

{ELIMINAR 1 (lista desordenada, sin repetición, no se sabe si existe) - LISTAS}

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, sin repetición, se sabe que existe) - LISTAS}

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

{ELIMINAR 3 (lista desordenada, con repetición, se sabe que existen, al menos, dos elementos a
eliminar) - LISTAS}

procedure eliminar3(var lista: t_lista; var elimino: boolean; valor: integer);
var
  anterior, actual: t_lista;
begin
  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
      begin
        lista:=lista^.sig;
        anterior:=lista;
      end
      else
        anterior^.sig:=actual^.sig;
      dispose(actual);
      actual:=anterior;
      elimino:=true;
    end;
  end;
end;

{ELIMINAR 4 (lista ordenada, sin repetición, no se sabe si existe) - LISTAS}

procedure eliminar4(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 5 (lista ordenada, sin repetición, se sabe que existe) - LISTAS}

procedure eliminar5(var lista: t_lista; var elimino: boolean; valor: integer);
var
  anterior, actual: t_lista;
begin
  actual:=lista;
  while (actual^.ele.ele1<valor) do
  begin
    anterior:=actual;
    actual:=actual^.sig;
  end;
  if (actual=lista) then
    lista:=lista^.sig
  else
    anterior^.sig:=actual^.sig;
  dispose(actual);
  elimino:=true;
end;
```

```
{ELIMINAR 6 (lista ordenada, con repetición, se sabe que existen, al menos, dos elementos a eliminar) - LISTAS}

procedure eliminar6(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
    if (actual^.ele.ele1<valor) then
    begin
      anterior:=actual;
      actual:=actual^.sig;
    end
    else
    begin
      if (actual=lista) then
      begin
        lista:=lista^.sig;
        anterior:=lista;
      end
      else
        anterior^.sig:=actual^.sig;
      dispose(actual);
      actual:=anterior;
      elimino:=true;
    end;
  end;
end;
```