**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;