Juan Menduiña

Trabajo Práctico N° 5:
Módulo Imperativo (Adicionales).

Ejercicio 1.

El administrador de un edificio de oficinas cuenta, en papel, con la información del pago de las expensas de dichas oficinas. Implementar un programa con:

(a) *Un módulo que retorne un vector, sin orden, con, a lo sumo, las 300 oficinas que administra. Se debe leer, para cada oficina, el código de identificación, DNI del propietario y valor de la expensa. La lectura finaliza cuando llega el código de identificación -1.*

(b) *Un módulo que reciba el vector retornado en (a) y retorne dicho vector ordenado por código de identificación de la oficina. Ordenar el vector aplicando uno de los métodos vistos en la cursada.*

(c) *Un módulo que realice una búsqueda dicotómica. Este módulo debe recibir el vector generado en (b) y un código de identificación de oficina. En el caso de encontrarlo, debe retornar la posición del vector donde se encuentra y, en caso contrario, debe retornar 0. Luego, el programa debe informar el DNI del propietario o un cartel indicando que no se encontró la oficina.*

(d) *Un módulo recursivo que retorne el monto total de las expensas.*

```
program TP5_E1;
{$codepage UTF8}
uses crt;
const
  oficinas_min=1; oficinas_max=300; codigo_salida=-1;
type
  t_oficinas=oficinas_min..oficinas_max;
  t_registro_oficinas=record
    codigo: int16;
    dni: int32;
    valor: real;
  end;
  t_vector_oficinas=array[t_oficinas] of t_registro_oficinas;
procedure inicializar_vector_oficinas(var vector_oficinas: t_vector_oficinas);
var
  i: int16;
begin
  for i:= oficinas_min to oficinas_max do
  begin
    vector_oficinas[i].codigo:=0;
    vector_oficinas[i].dni:=0;
    vector_oficinas[i].valor:=0;
  end;
end;
procedure leer_registro_oficinas(var registro_oficinas: t_registro_oficinas);
begin
  textcolor(green); write('Introducir código de identificación de la oficina: ');
textcolor(yellow); readln(registro_oficinas.codigo);
  if (registro_oficinas.codigo<>codigo_salida) then
  begin
```

```
    textcolor(green); write('Introducir DNI del propietario de la oficina: ');
textcolor(yellow); readln(registro_oficinas.dni);
    textcolor(green); write('Introducir valor de la expensa de la oficina: ');
textcolor(yellow); readln(registro_oficinas.valor);
  end;
end;
procedure cargar_vector_oficinas(var vector_oficinas: t_vector_oficinas; var oficinas:
int16);
var
  registro_oficinas: t_registro_oficinas;
begin
  leer_registro_oficinas(registro_oficinas);
  while ((registro_oficinas.codigo<>codigo_salida) and (oficinas<oficinas_max)) do
  begin
    oficinas:=oficinas+1;
    vector_oficinas[oficinas]:=registro_oficinas;
    leer_registro_oficinas(registro_oficinas);
  end;
end;
procedure ordenar_vector_oficinas(var vector_oficinas: t_vector_oficinas; oficinas: int16);
var
  i, j, k, item: int16;
begin
  for i:= 1 to (oficinas-1) do
  begin
    k:=i;
    for j:= (i+1) to oficinas do
      if (vector_oficinas[j].codigo<vector_oficinas[k].codigo) then
        k:=j;
    item:=vector_oficinas[k].codigo;
    vector_oficinas[k].codigo:=vector_oficinas[i].codigo;
    vector_oficinas[i].codigo:=item;
  end;
end;
function buscar_vector_oficinas(vector_oficinas: t_vector_oficinas; oficinas, codigo:
int16): int16;
var
  pri, ult, medio: int16;
begin
  pri:=1; ult:=oficinas; medio:=(pri+ult) div 2;
  while ((pri<=ult) and (vector_oficinas[medio].codigo<>codigo)) do
  begin
    if (vector_oficinas[medio].codigo>codigo) then
      ult:=medio-1
    else
      pri:=medio+1;
    medio:=(pri+ult) div 2;
  end;
  if (pri<=ult) then
    buscar_vector_oficinas:=medio
  else
    buscar_vector_oficinas:=0;
end;
function sumar_expensas(vector_oficinas: t_vector_oficinas; oficinas: int16): real;
var
  suma: real;
begin
  suma:=0;
  if (oficinas<>0) then
    suma:=vector_oficinas[oficinas].valor+sumar_expensas(vector_oficinas,oficinas-1);
  sumar_expensas:=suma;
end;
var
  vector_oficinas: t_vector_oficinas;
  oficinas, codigo, pos: int16;
  expensas: real;
```

```
begin
  inicializar_vector_oficinas(vector_oficinas); oficinas:=0;
  cargar_vector_oficinas(vector_oficinas,oficinas);
  ordenar_vector_oficinas(vector_oficinas,oficinas);
  textcolor(green); write('Introducir código de identificación de la oficina que se desea
buscar: '); textcolor(yellow); readln(codigo);
  pos:=buscar_vector_oficinas(vector_oficinas,oficinas,codigo);
  if (pos<>0) then
  begin
    textcolor(green); write('El DNI del propietario con código de identificación ');
textcolor(red); write(codigo); textcolor(green); write(' es '); textcolor(red);
writeln(vector_oficinas[pos].dni);
  end
  else
  begin
    textcolor(red); writeln('No se encontró la oficina');
  end;
  expensas:=sumar_expensas(vector_oficinas,oficinas);
  textcolor(green); write('El monto total de las expensas es $'); textcolor(red);
write(expensas:0:2);
end.
```

Ejercicio 2.

Una agencia dedicada a la venta de autos ha organizado su stock y dispone, en papel, de la información de los autos en venta. Implementar un programa que:

(a) *Lea la información de los autos (patente, año de fabricación (2010 .. 2018), marca y modelo) y los almacene en dos estructuras de datos:*

> **(i)** *Una estructura eficiente para la búsqueda por patente.*
> **(ii)** *Una estructura eficiente para la búsqueda por marca. Para cada marca, se deben almacenar todos juntos los autos pertenecientes a ella.*

(b) *Invoque a un módulo que reciba la estructura generado en (a) (i) y una marca y retorne la cantidad de autos de dicha marca que posee la agencia.*

(c) *Invoque a un módulo que reciba la estructura generado en (a) (ii) y una marca y retorne la cantidad de autos de dicha marca que posee la agencia.*

(d) *Invoque a un módulo que reciba el árbol generado en (a) (i) y retorne una estructura con la información de los autos agrupados por año de fabricación.*

(e) *Invoque a un módulo que reciba el árbol generado en (a) (i) y una patente y devuelva el modelo del auto con dicha patente.*

(f) *Invoque a un módulo que reciba el árbol generado en (a) (ii) y una patente y devuelva el modelo del auto con dicha patente.*

```
program TP5_E2;
{$codepage UTF8}
uses crt;
const
  anio_min=2010; anio_max=2018; patente_salida='-1';
type
  t_anios=anio_min..anio_max;
  t_registro_auto=record
    patente: string;
    anio: t_anios;
    marca: string;
    modelo: string;
  end;
  t_abb_i=^t_nodo_abb_i;
  t_nodo_abb_i=record
    ele: t_registro_auto;
    hi: t_abb_i;
    hd: t_abb_i;
  end;
  t_lista_autos=^t_nodo_marca;
  t_nodo_marca=record
    ele: t_registro_auto;
    sig: t_lista_autos;
  end;
  t_abb_ii=^t_nodo_abb_ii;
  t_nodo_abb_ii=record
    ele: t_lista_autos;
    hi: t_abb_ii;
    hd: t_abb_ii;
```

```
  end;
  t_vector_autos=array[t_anios] of t_lista_autos;
procedure inicializar_vector_autos(var vector_autos: t_vector_autos);
var
  i: t_anios;
begin
  for i:= anio_min to anio_max do
    vector_autos[i]:=nil;
end;
procedure leer_registro_auto(var registro_auto: t_registro_auto);
begin
  textcolor(green); write('Introducir patente del auto en venta: '); textcolor(yellow);
readln(registro_auto.patente);
  if (registro_auto.patente<>patente_salida) then
  begin
    textcolor(green); write('Introducir año de fabricación del auto en venta: ');
textcolor(yellow); readln(registro_auto.anio);
    textcolor(green); write('Introducir marca del auto en venta: '); textcolor(yellow);
readln(registro_auto.marca);
    textcolor(green); write('Introducir modelo del auto en venta: '); textcolor(yellow);
readln(registro_auto.modelo);
  end;
end;
procedure agregar_abb_i(var abb_i: t_abb_i; registro_auto: t_registro_auto);
begin
  if (abb_i=nil) then
  begin
    new(abb_i);
    abb_i^.ele:=registro_auto;
    abb_i^.hi:=nil;
    abb_i^.hd:=nil;
  end
  else
    if (registro_auto.patente<=abb_i^.ele.patente) then
      agregar_abb_i(abb_i^.hi,registro_auto)
    else
      agregar_abb_i(abb_i^.hd,registro_auto);
end;
procedure agregar_adelante_lista_autos(var lista_autos: t_lista_autos; registro_auto:
t_registro_auto);
var
  nuevo: t_lista_autos;
begin
  new(nuevo);
  nuevo^.ele:=registro_auto;
  nuevo^.sig:=lista_autos;
  lista_autos:=nuevo;
end;
procedure agregar_abb_ii(var abb_ii: t_abb_ii; registro_auto: t_registro_auto);
begin
  if (abb_ii=nil) then
  begin
    new(abb_ii);
    abb_ii^.ele:=nil;
    agregar_adelante_lista_autos(abb_ii^.ele,registro_auto);
    abb_ii^.hi:=nil;
    abb_ii^.hd:=nil;
  end
  else
    if (registro_auto.marca=abb_ii^.ele^.ele.marca) then
      agregar_adelante_lista_autos(abb_ii^.ele,registro_auto)
    else
      if (registro_auto.marca<abb_ii^.ele^.ele.marca) then
        agregar_abb_ii(abb_ii^.hi,registro_auto)
      else
        agregar_abb_ii(abb_ii^.hd,registro_auto);
```

```
end;
procedure cargar_abbs(var abb_i: t_abb_i; var abb_ii: t_abb_ii);
var
  registro_auto: t_registro_auto;
begin
  leer_registro_auto(registro_auto);
  while (registro_auto.patente<>patente_salida) do
  begin
    agregar_abb_i(abb_i,registro_auto);
    agregar_abb_ii(abb_ii,registro_auto);
    leer_registro_auto(registro_auto);
  end;
end;
function inciso_b(abb_i: t_abb_i; marca: string): int16;
var
  cantidad: int16;
begin
  cantidad:=0;
  if (abb_i<>nil) then
  begin
    if (abb_i^.ele.marca=marca) then
      cantidad:=inciso_b(abb_i^.hi,marca)+inciso_b(abb_i^.hd,marca)+1
    else
      cantidad:=inciso_b(abb_i^.hi,marca)+inciso_b(abb_i^.hd,marca);
  end;
  inciso_b:=cantidad;
end;
function recorrer_lista_autos_1(lista_autos: t_lista_autos): int16;
var
  cantidad: int16;
begin
  cantidad:=0;
  while (lista_autos<>nil) do
  begin
    cantidad:=cantidad+1;
    lista_autos:=lista_autos^.sig;
  end;
  recorrer_lista_autos_1:=cantidad;
end;
function inciso_c(abb_ii: t_abb_ii; marca: string): int16;
var
  cantidad: int16;
begin
  cantidad:=0;
  if (abb_ii<>nil) then
  begin
    if (abb_ii^.ele^.ele.marca=marca) then
      cantidad:=recorrer_lista_autos_1(abb_ii^.ele)
    else
      if (marca<=abb_ii^.ele^.ele.marca) then
        cantidad:=inciso_c(abb_ii^.hi,marca)
      else
        cantidad:=inciso_c(abb_ii^.hd,marca);
  end;
  inciso_c:=cantidad;
end;
procedure inciso_d(var vector_autos: t_vector_autos; abb_i: t_abb_i);
begin
  if (abb_i<>nil) then
  begin
    inciso_d(vector_autos,abb_i^.hi);
    agregar_adelante_lista_autos(vector_autos[abb_i^.ele.anio],abb_i^.ele);
    inciso_d(vector_autos,abb_i^.hd);
  end;
end;
function inciso_e(abb_i: t_abb_i; patente: string): string;
```

```
var
  modelo: string;
begin
  modelo:='No se encontró la patente';
  if (abb_i<>nil) then
  begin
    if (abb_i^.ele.patente=patente) then
      modelo:=abb_i^.ele.modelo
    else
      if (patente<=abb_i^.ele.patente) then
        modelo:=inciso_e(abb_i^.hi,patente)
      else
        modelo:=inciso_e(abb_i^.hd,patente);
  end;
  inciso_e:=modelo;
end;
function recorrer_lista_autos_2(lista_autos: t_lista_autos; patente: string): t_lista_autos;
begin
  while ((lista_autos<>nil) and (lista_autos^.ele.patente<>patente)) do
    lista_autos:=lista_autos^.sig;
  recorrer_lista_autos_2:=lista_autos;
end;
function inciso_f(abb_ii: t_abb_ii; patente: string): string;
var
  lista_autos: t_lista_autos;
  modelo: string;
begin
  lista_autos:=nil; modelo:='No se encontró la patente';
  if (abb_ii<>nil) then
  begin
    lista_autos:=recorrer_lista_autos_2(abb_ii^.ele,patente);
    if (lista_autos<>nil) then
      modelo:=abb_ii^.ele^.ele.modelo
    else
    begin
      modelo:=inciso_f(abb_ii^.hi,patente);
      if (modelo='No se encontró la patente') then
        modelo:=inciso_f(abb_ii^.hd,patente);
    end;
  end;
  inciso_f:=modelo;
end;
var
  vector_autos: t_vector_autos;
  abb_i: t_abb_i;
  abb_ii: t_abb_ii;
  autos_b, autos_c: int16;
  marca, patente, modelo_e, modelo_f: string;
begin
  abb_i:=nil; abb_ii:=nil; inicializar_vector_autos(vector_autos);
  cargar_abbs(abb_i,abb_ii);
  textcolor(green); write('Introducir marca del auto en venta que se desea buscar en los
árboles: '); textcolor(yellow); readln(marca);
  autos_b:=inciso_b(abb_i,marca);
  textcolor(green); write('La cantidad de autos de la marca '); textcolor(red);
write(marca); textcolor(green); write(' en el árbol abb_i es '); textcolor(red);
writeln(autos_b);
  autos_c:=inciso_c(abb_ii,marca);
  textcolor(green); write('La cantidad de autos de la marca '); textcolor(red);
write(marca); textcolor(green); write(' en el árbol abb_ii es '); textcolor(red);
writeln(autos_c);
  inciso_d(vector_autos,abb_i);
  textcolor(green); write('Introducir patente del auto en venta que se desea buscar en los
árboles: '); textcolor(yellow); readln(patente);
  modelo_e:=inciso_e(abb_i,patente);
```

```
  textcolor(green); write('El modelo del auto de la patente '); textcolor(red);
write(patente); textcolor(green); write(' en el árbol abb_i es '); textcolor(red);
writeln(modelo_e);
  modelo_f:=inciso_f(abb_ii,patente);
  textcolor(green); write('El modelo del auto de la patente '); textcolor(red);
write(patente); textcolor(green); write(' en el árbol abb_ii es '); textcolor(red);
write(modelo_f);
end.
```

Ejercicio 3.

Un supermercado requiere el procesamiento de sus productos. De cada producto, se conoce código, rubro (1..10), stock y precio unitario. Se pide:

(a) *Generar una estructura adecuada que permita agrupar los productos por rubro. A su vez, para cada rubro, se requiere que la búsqueda de un producto por código sea lo más eficiente posible. La lectura finaliza con el código de producto igual a -1.*

(b) *Implementar un módulo que reciba la estructura generada en (a), un rubro y un código de producto y retorne si dicho código existe o no para ese rubro.*

(c) *Implementar un módulo que reciba la estructura generada en (a) y retorne, para cada rubro, el código y stock del producto con mayor código.*

(d) *Implementar un módulo que reciba la estructura generada en (a), dos códigos y retorne, para cada rubro, la cantidad de productos con códigos entre los dos valores ingresados.*

```
program TP5_E3;
{$codepage UTF8}
uses crt;
const
  rubro_min=1; rubro_max=10; codigo_salida=-1;
type
  t_rubros=rubro_min..rubro_max;
  t_registro_producto=record
    codigo: int16;
    rubro: t_rubros;
    stock: int16;
    precio: real;
  end;
  t_abb=^t_nodo_abb;
  t_nodo_abb=record
    ele: t_registro_producto;
    hi: t_abb;
    hd: t_abb;
  end;
  t_vector_rubros=array[t_rubros] of t_abb;
  t_registro_c=record
    codigo: int16;
    stock: int16;
  end;
  t_vector_c=array[t_rubros] of t_registro_c;
  t_vector_d=array[t_rubros] of int32;
procedure inicializar_vector_rubros(var vector_rubros: t_vector_rubros);
var
  i: t_rubros;
begin
  for i:= rubro_min to rubro_max do
    vector_rubros[i]:=nil;
end;
procedure inicializar_vector_c(var vector_c: t_vector_c);
var
  i: t_rubros;
begin
  for i:= rubro_min to rubro_max do
  begin
    vector_c[i].codigo:=0;
```

```
    vector_c[i].stock:=0;
  end;
end;
procedure inicializar_vector_d(var vector_d: t_vector_d);
var
  i: t_rubros;
begin
  for i:= rubro_min to rubro_max do
    vector_d[i]:=0;
end;
procedure leer_registro_producto(var registro_producto: t_registro_producto);
begin
  textcolor(green); write('Introducir código del producto: '); textcolor(yellow);
readln(registro_producto.codigo);
  if (registro_producto.codigo<>codigo_salida) then
  begin
    textcolor(green); write('Introducir rubro del producto: '); textcolor(yellow);
readln(registro_producto.rubro);
    textcolor(green); write('Introducir stock del producto: '); textcolor(yellow);
readln(registro_producto.stock);
    textcolor(green); write('Introducir precio del producto: '); textcolor(yellow);
readln(registro_producto.precio);
  end;
end;
procedure agregar_abb(var abb: t_abb; registro_producto: t_registro_producto);
begin
  if (abb=nil) then
  begin
    new(abb);
    abb^.ele:=registro_producto;
    abb^.hi:=nil;
    abb^.hd:=nil;
  end
  else
    if (registro_producto.codigo<=abb^.ele.codigo) then
      agregar_abb(abb^.hi,registro_producto)
    else
      agregar_abb(abb^.hd,registro_producto);
end;
procedure cargar_vector_rubros(var vector_rubros: t_vector_rubros);
var
  registro_producto: t_registro_producto;
begin
  leer_registro_producto(registro_producto);
  while (registro_producto.codigo<>codigo_salida) do
  begin
    agregar_abb(vector_rubros[registro_producto.rubro],registro_producto);
    leer_registro_producto(registro_producto);
  end;
end;
function buscar_abb(abb: t_abb; codigo: int16): boolean;
begin
  if (abb=nil) then
    buscar_abb:=false
  else
    if (abb^.ele.codigo=codigo) then
      buscar_abb:=true
    else
      if (codigo<=abb^.ele.codigo) then
        buscar_abb:=buscar_abb(abb^.hi,codigo)
      else
        buscar_abb:=buscar_abb(abb^.hd,codigo);
end;
function inciso_b(vector_rubros: t_vector_rubros; rubro: t_rubros; codigo: int16): boolean;
begin
  inciso_b:=buscar_abb(vector_rubros[rubro],codigo);
```

```
end;
function buscar_maximo_abb(abb: t_abb): t_abb;
begin
  if (abb^.hd=nil) then
    buscar_maximo_abb:=abb
  else
    buscar_maximo_abb:=buscar_maximo_abb(abb^.hd);
end;
procedure inciso_c(var vector_c: t_vector_c; vector_rubros: t_vector_rubros);
var
  abb_max: t_abb;
  i: t_rubros;
begin
  abb_max:=nil;
  for i:= rubro_min to rubro_max do
  begin
    if (vector_rubros[i]<>nil) then
    begin
      abb_max:=buscar_maximo_abb(vector_rubros[i]);
      vector_c[i].codigo:=abb_max^.ele.codigo;
      vector_c[i].stock:=abb_max^.ele.stock;
    end;
  end;
end;
function contar_abb(abb: t_abb; codigo1, codigo2: int16): int32;
var
  cantidad: int32;
begin
  cantidad:=0;
  if (abb<>nil) then
  begin
    if ((abb^.ele.codigo>=codigo1) and (abb^.ele.codigo<=codigo2)) then
      cantidad:=contar_abb(abb^.hi,codigo1,codigo2)+contar_abb(abb^.hd,codigo1,codigo2)+1
    else
      if (codigo2<=abb^.ele.codigo) then
        cantidad:=cantidad+contar_abb(abb^.hi,codigo1,codigo2)
      else
        cantidad:=cantidad+contar_abb(abb^.hd,codigo1,codigo2);
  end;
  contar_abb:=cantidad;
end;
procedure inciso_d(var vector_d: t_vector_d; vector_rubros: t_vector_rubros; codigo1,
codigo2: int16);
var
  i: t_rubros;
begin
  for i:= rubro_min to rubro_max do
    if (vector_rubros[i]<>nil) then
      vector_d[i]:=contar_abb(vector_rubros[i],codigo1,codigo2);
end;
var
  vector_rubros: t_vector_rubros;
  vector_c: t_vector_c;
  vector_d: t_vector_d;
  rubro: t_rubros;
  codigo, codigo1, codigo2: int16;
  existe: boolean;
begin
  inicializar_vector_rubros(vector_rubros); inicializar_vector_c(vector_c);
inicializar_vector_d(vector_d); existe:=false;
  cargar_vector_rubros(vector_rubros);
  textcolor(green); write('Introducir rubro que se desea buscar en el vector de árboles: ');
textcolor(yellow); readln(rubro);
  textcolor(green); write('Introducir código que se desea buscar en el vector de árboles:
'); textcolor(yellow); readln(codigo);
  if (vector_rubros[rubro]<>nil) then
```

```
    existe:=inciso_b(vector_rubros,rubro,codigo);
  if (existe=true) then
  begin
    textcolor(green); write('El código '); textcolor(red); write(codigo); textcolor(green);
write(', para el rubro '); textcolor(red); write(rubro); textcolor(green); write(', ');
textcolor(red); writeln('EXISTE');
  end
  else
  begin
    textcolor(green); write('El código '); textcolor(red); write(codigo); textcolor(green);
write(', para el rubro '); textcolor(red); write(rubro); textcolor(green); write(', ');
textcolor(red); writeln('NO EXISTE');
  end;
  inciso_c(vector_c,vector_rubros);
  textcolor(green); writeln('Para buscar la cantidad de productos con códigos entre los
siguientes valores ingresados: ');
  textcolor(green); write('Introducir código 1: '); textcolor(yellow); readln(codigo1);
  textcolor(green); write('Introducir código 2: '); textcolor(yellow); read(codigo2);
  inciso_d(vector_d,vector_rubros,codigo1,codigo2);
end.
```

Ejercicio 4.

Una oficina requiere el procesamiento de los reclamos de las personas. De cada reclamo, se lee código, DNI de la persona, año y tipo de reclamo. La lectura finaliza con el código de igual a -1. Se pide:

(a) *Un módulo que retorne estructura adecuada para la búsqueda por DNI. Para cada DNI, se deben tener almacenados cada reclamo y la cantidad total de reclamos que realizó.*

(b) *Un módulo que reciba la estructura generada en (a) y un DNI y retorne la cantidad de reclamos efectuados por ese DNI.*

(c) *Un módulo que reciba la estructura generada en (a) y dos DNI y retorne la cantidad de reclamos efectuados por todos los DNI comprendidos entre los dos DNI recibidos.*

(d) *Un módulo que reciba la estructura generada en (a) y un año y retorne los códigos de los reclamos realizados en el año recibido.*

```
program TP5_E4;
{$codepage UTF8}
uses crt;
const
  codigo_salida=-1;
type
  t_registro_reclamo=record
    codigo: int16;
    dni: int32;
    anio: int16;
    tipo: string;
  end;
  t_lista_reclamos=^t_nodo_reclamos;
  t_nodo_reclamos=record
    ele: t_registro_reclamo;
    sig: t_lista_reclamos;
  end;
  t_registro_dni=record
    dni: int32;
    reclamos: t_lista_reclamos;
    cantidad: int16;
  end;
  t_abb=^t_nodo_abb;
  t_nodo_abb=record
    ele: t_registro_dni;
    hi: t_abb;
    hd: t_abb;
  end;
  t_lista_codigos=^t_nodo_codigos;
  t_nodo_codigos=record
    ele: int16;
    sig: t_lista_codigos;
  end;
procedure leer_registro_reclamo(var registro_reclamo: t_registro_reclamo);
begin
  textcolor(green); write('Introducir código del reclamo: '); textcolor(yellow);
readln(registro_reclamo.codigo);
  if (registro_reclamo.codigo<>codigo_salida) then
  begin
```

```
    textcolor(green); write('Introducir DNI de la persona del reclamo: ');
textcolor(yellow); readln(registro_reclamo.dni);
    textcolor(green); write('Introducir año del reclamo: '); textcolor(yellow);
readln(registro_reclamo.anio);
    textcolor(green); write('Introducir tipo del reclamo: '); textcolor(yellow);
readln(registro_reclamo.tipo);
  end;
end;
procedure agregar_adelante_lista_reclamos(var lista_reclamos: t_lista_reclamos;
registro_reclamo: t_registro_reclamo);
var
  nuevo: t_lista_reclamos;
begin
  new(nuevo);
  nuevo^.ele:=registro_reclamo;
  nuevo^.sig:=lista_reclamos;
  lista_reclamos:=nuevo;
end;
procedure agregar_abb(var abb: t_abb; registro_reclamo: t_registro_reclamo);
var
  registro_dni: t_registro_dni;
begin
  if (abb=nil) then
  begin
    registro_dni.dni:=registro_reclamo.dni;
    registro_dni.reclamos:=nil;
    agregar_adelante_lista_reclamos(registro_dni.reclamos,registro_reclamo);
    registro_dni.cantidad:=1;
    new(abb);
    abb^.ele:=registro_dni;
    abb^.hi:=nil;
    abb^.hd:=nil;
  end
  else
    if (registro_reclamo.dni=abb^.ele.dni) then
    begin
      agregar_adelante_lista_reclamos(abb^.ele.reclamos,registro_reclamo);
      abb^.ele.cantidad:=abb^.ele.cantidad+1;
    end
    else
      if (registro_reclamo.dni<abb^.ele.dni) then
        agregar_abb(abb^.hi,registro_reclamo)
      else
        agregar_abb(abb^.hd,registro_reclamo);
end;
procedure cargar_abb(var abb: t_abb);
var
  registro_reclamo: t_registro_reclamo;
begin
  leer_registro_reclamo(registro_reclamo);
  while (registro_reclamo.codigo<>codigo_salida) do
  begin
    agregar_abb(abb,registro_reclamo);
    leer_registro_reclamo(registro_reclamo);
  end;
end;
function inciso_b(abb: t_abb; dni: int32): int32;
begin
  if (abb=nil) then
    inciso_b:=0
  else
    if (abb^.ele.dni=dni) then
      inciso_b:=abb^.ele.cantidad
    else
      if (dni<=abb^.ele.dni) then
        inciso_b:=inciso_b(abb^.hi,dni)
```

```
      else
        inciso_b:=inciso_b(abb^.hd,dni);
end;
function inciso_c(abb: t_abb; dni1, dni2: int32): int32;
var
  cantidad: int32;
begin
  cantidad:=0;
  if (abb<>nil) then
  begin
    if ((abb^.ele.dni>=dni1) and (abb^.ele.dni<=dni2)) then
      cantidad:=inciso_c(abb^.hi,dni1,dni2)+inciso_c(abb^.hd,dni1,dni2)+1
    else
      if (dni2<=abb^.ele.dni) then
        cantidad:=cantidad+inciso_c(abb^.hi,dni1,dni2)
      else
        cantidad:=cantidad+inciso_c(abb^.hd,dni1,dni2);
  end;
  inciso_c:=cantidad;
end;
procedure agregar_adelante_lista_codigos(var lista_codigos: t_lista_codigos; codigo: int16);
var
  nuevo: t_lista_codigos;
begin
  new(nuevo);
  nuevo^.ele:=codigo;
  nuevo^.sig:=lista_codigos;
  lista_codigos:=nuevo;
end;
procedure recorrer_lista_reclamos(var lista_codigos: t_lista_codigos; lista_reclamos:
t_lista_reclamos; anio: int16);
begin
  while (lista_reclamos<>nil) do
  begin
    if (lista_reclamos^.ele.anio=anio) then
      agregar_adelante_lista_codigos(lista_codigos,lista_reclamos^.ele.codigo);
    lista_reclamos:=lista_reclamos^.sig;
  end;
end;
procedure inciso_d(var lista_codigos: t_lista_codigos; abb: t_abb; anio: int16);
begin
  if (abb<>nil) then
  begin
    inciso_d(lista_codigos,abb^.hi,anio);
    recorrer_lista_reclamos(lista_codigos,abb^.ele.reclamos,anio);
    inciso_d(lista_codigos,abb^.hd,anio);
  end;
end;
var
  lista_codigos: t_lista_codigos;
  abb: t_abb;
  anio: int16;
  dni, dni1, dni2, reclamos_b, reclamos_c: int32;
begin
  abb:=nil; lista_codigos:=nil; reclamos_b:=0; reclamos_c:=0;
  cargar_abb(abb);
  textcolor(green); write('Introducir DNI para el cual se desea buscar la cantidad de
reclamos: '); textcolor(yellow); readln(dni);
  reclamos_b:=inciso_b(abb,dni);
  textcolor(green); write('La cantidad de reclamos del DNI '); textcolor(red); write(dni);
textcolor(green); write(' en el árbol abb es '); textcolor(red); writeln(reclamos_b);
  textcolor(green); writeln('Para buscar la cantidad de reclamos con DNIs entre los
siguientes valores ingresados: ');
  textcolor(green); write('Introducir DNI 1: '); textcolor(yellow); readln(dni1);
  textcolor(green); write('Introducir DNI 2: '); textcolor(yellow); readln(dni2);
  reclamos_c:=inciso_c(abb,dni1,dni2);
```

```
  textcolor(green); write('La cantidad de reclamos entre los DNI '); textcolor(red);
write(dni1); textcolor(green); write(' y '); textcolor(red); write(dni2); textcolor(green);
write(' en el árbol abb es '); textcolor(red); writeln(reclamos_c);
  textcolor(green); write('Introducir año para el cual se desea retornar los códigos de los
reclamos realizados: '); textcolor(yellow); read(anio);
  inciso_d(lista_codigos,abb,anio);
end.
```

Ejercicio 5.

Realizar el inciso (a) del ejercicio anterior, pero sabiendo que todos los reclamos de un mismo DNI se leen de forma consecutiva (no significa que vengan ordenados los DNI).

```
program TP5_E5;
{$codepage UTF8}
uses crt;
const
  codigo_salida=-1;
type
  t_registro_reclamo=record
    codigo: int16;
    dni: int32;
    anio: int16;
    tipo: string;
  end;
  t_lista_reclamos=^t_nodo_reclamos;
  t_nodo_reclamos=record
    ele: t_registro_reclamo;
    sig: t_lista_reclamos;
  end;
  t_registro_dni=record
    dni: int32;
    reclamos: t_lista_reclamos;
    cantidad: int16;
  end;
  t_abb=^t_nodo_abb;
  t_nodo_abb=record
    ele: t_registro_dni;
    hi: t_abb;
    hd: t_abb;
  end;
procedure leer_registro_reclamo(var registro_reclamo: t_registro_reclamo);
begin
  textcolor(green); write('Introducir código del reclamo: '); textcolor(yellow);
readln(registro_reclamo.codigo);
  if (registro_reclamo.codigo<>codigo_salida) then
  begin
    textcolor(green); write('Introducir DNI de la persona del reclamo: ');
textcolor(yellow); readln(registro_reclamo.dni);
    textcolor(green); write('Introducir año del reclamo: '); textcolor(yellow);
readln(registro_reclamo.anio);
    textcolor(green); write('Introducir tipo del reclamo: '); textcolor(yellow);
readln(registro_reclamo.tipo);
  end;
end;
procedure agregar_adelante_lista_reclamos(var lista_reclamos: t_lista_reclamos;
registro_reclamo: t_registro_reclamo);
var
  nuevo: t_lista_reclamos;
begin
  new(nuevo);
  nuevo^.ele:=registro_reclamo;
  nuevo^.sig:=lista_reclamos;
  lista_reclamos:=nuevo;
end;
procedure agregar_abb(var abb: t_abb; registro_reclamo: t_registro_reclamo);
var
  registro_dni: t_registro_dni;
begin
  if (abb=nil) then
  begin
```

```
    registro_dni.dni:=registro_reclamo.dni;
    registro_dni.reclamos:=nil;
    agregar_adelante_lista_reclamos(registro_dni.reclamos,registro_reclamo);
    registro_dni.cantidad:=1;
    new(abb);
    abb^.ele:=registro_dni;
    abb^.hi:=nil;
    abb^.hd:=nil;
  end
  else
    if (registro_reclamo.dni<abb^.ele.dni) then
      agregar_abb(abb^.hi,registro_reclamo)
    else
      if (registro_reclamo.dni>abb^.ele.dni) then
        agregar_abb(abb^.hd,registro_reclamo);
end;
procedure cargar_abb(var abb: t_abb; var registro_reclamo: t_registro_reclamo);
var
  dni_actual: int32;
begin
  leer_registro_reclamo(registro_reclamo);
  while (registro_reclamo.codigo<>codigo_salida) do
  begin
    dni_actual:=registro_reclamo.dni;
    agregar_abb(abb,registro_reclamo);
    leer_registro_reclamo(registro_reclamo);
    while ((registro_reclamo.codigo<>codigo_salida) and (registro_reclamo.dni=dni_actual))
do
    begin
      agregar_adelante_lista_reclamos(abb^.ele.reclamos,registro_reclamo);
      abb^.ele.cantidad:=abb^.ele.cantidad+1;
      leer_registro_reclamo(registro_reclamo);
    end;
  end;
end;
var
  registro_reclamo: t_registro_reclamo;
  abb: t_abb;
begin
  abb:=nil;
  cargar_abb(abb,registro_reclamo);
end.
```