**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.