

Trabajo Práctico N° 0: **Módulo Imperativo (Práctica Inicial).**

Ejercicio 1.

Implementar un programa que procese la información de los alumnos de la Facultad de Informática.

(a) Implementar un módulo que lea y retorne, en una estructura adecuada, la información de todos los alumnos. De cada alumno, se lee su apellido, número de alumno, año de ingreso, cantidad de materias aprobadas (a lo sumo, 36) y nota obtenida (sin contar los aplazos) en cada una de las materias aprobadas. La lectura finaliza cuando se ingresa el número de alumno 11.111, el cual debe procesarse.

(b) Implementar un módulo que reciba la estructura generada en el inciso (a) y retorne número de alumno y promedio de cada alumno.

```
program TP0_E1ab;
{$codepage UTF8}
uses crt;
const
  anio_ini=2000; anio_fin=2023;
  materias_total=36;
  nota_ini=4; nota_fin=10;
  numero_salida=11111;
type
  t_materia=1..materias_total;
  t_nota=nota_ini..nota_fin;
  t_vector_notas=array[t_materia] of t_nota;
  t_registro_alumno1=record
    apellido: string;
    numero: int16;
    anio_ingreso: int16;
    materias_aprobadas: int8;
    notas: t_vector_notas;
  end;
  t_registro_alumno2=record
    numero: int32;
    promedio: real;
  end;
  t_lista_alumnos1=^t_nodo_alumnos1;
  t_nodo_alumnos1=record
    ele: t_registro_alumno1;
    sig: t_lista_alumnos1;
  end;
  t_lista_alumnos2=^t_nodo_alumnos2;
  t_nodo_alumnos2=record
    ele: t_registro_alumno2;
    sig: t_lista_alumnos2;
  end;
function random_string(length: int8): string;
var
  i: int8;
  string_aux: string;
begin
  string_aux:='';
  for i:= 1 to length do
    string_aux:=string_aux+chr(ord('A')+random(26));
```

```

    random_string:=string_aux;
end;
procedure leer_alumno(var registro_alumno1: t_registro_alumno1);
var
    i: int8;
begin
    registro_alumno1.apellido:=random_string(5+random(6));
    i:=random(100);
    if (i=0) then
        registro_alumno1.numero:=numero_salida
    else
        registro_alumno1.numero:=1+random(high(int16));
        registro_alumno1.anio_ingreso:=anio_ini+random(anio_fin-anio_ini+1);
        registro_alumno1.materias_aprobadas:=random(materias_total+1);
        for i:= 1 to registro_alumno1.materias_aprobadas do
            registro_alumno1.notas[i]:=nota_ini+random(nota_fin-nota_ini+1);
        end;
    end;
procedure agregar_adelante_lista_alumnos1(var lista_alumnos1: t_lista_alumnos1;
registro_alumno1: t_registro_alumno1);
var
    nuevo: t_lista_alumnos1;
begin
    new(nuevo);
    nuevo^.ele:=registro_alumno1;
    nuevo^.sig:=lista_alumnos1;
    lista_alumnos1:=nuevo;
end;
procedure cargar_lista_alumnos1(var lista_alumnos1: t_lista_alumnos1);
var
    registro_alumno1: t_registro_alumno1;
begin
    repeat
        leer_alumno(registro_alumno1);
        agregar_adelante_lista_alumnos1(lista_alumnos1,registro_alumno1);
    until (registro_alumno1.numero=numero_salida);
end;
procedure imprimir_registro_alumno1(registro_alumno1: t_registro_alumno1; alumno: int16);
begin
    textcolor(green); write('El apellido del alumno '); textcolor(yellow); write(alumno);
textcolor(green); write(' es '); textcolor(red); writeln(registro_alumno1.apellido);
    textcolor(green); write('El número de alumno del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno1.numero);
    textcolor(green); write('El año de ingreso del alumno '); textcolor(yellow); write(alumno);
textcolor(green); write(' es '); textcolor(red); writeln(registro_alumno1.anio_ingreso);
    textcolor(green); write('La cantidad de materias aprobadas del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno1.materias_aprobadas);
end;
procedure imprimir_lista_alumnos1(lista_alumnos1: t_lista_alumnos1);
var
    i: int16;
begin
    i:=0;
    while (lista_alumnos1<>nil) do
        begin
            i:=i+1;
            textcolor(green); write('La información del alumno '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
            imprimir_registro_alumno1(lista_alumnos1^.ele,i);
            writeln();
            lista_alumnos1:=lista_alumnos1^.sig;
        end;
    end;
end;
procedure cargar_registro_alumno2(var registro_alumno2: t_registro_alumno2; registro_alumno1:
t_registro_alumno1);

```

```
var
  i: int8;
  suma: int16;
begin
  suma:=0;
  registro_alumno2.numero:=registro_alumno1.numero;
  if (registro_alumno1.materias_aprobadas<>0) then
  begin
    for i:= 1 to registro_alumno1.materias_aprobadas do
      suma:=suma+registro_alumno1.notas[i];
    registro_alumno2.promedio:=suma/registro_alumno1.materias_aprobadas;
  end
  else
    registro_alumno2.promedio:=suma;
  end;
procedure agregar_adelante_lista_alumnos2(var lista_alumnos2: t_lista_alumnos2;
registro_alumno2: t_registro_alumno2);
var
  nuevo: t_lista_alumnos2;
begin
  new(nuevo);
  nuevo^.ele:=registro_alumno2;
  nuevo^.sig:=lista_alumnos2;
  lista_alumnos2:=nuevo;
end;
procedure cargar_lista_alumnos2(var lista_alumnos2: t_lista_alumnos2; lista_alumnos1:
t_lista_alumnos1);
var
  registro_alumno2: t_registro_alumno2;
begin
  while (lista_alumnos1<>nil) do
  begin
    cargar_registro_alumno2(registro_alumno2,lista_alumnos1^.ele);
    agregar_adelante_lista_alumnos2(lista_alumnos2,registro_alumno2);
    lista_alumnos1:=lista_alumnos1^.sig;
  end;
end;
procedure imprimir_registro_alumno2(registro_alumno2: t_registro_alumno2; alumno: int16);
begin
  textcolor(green); write('El número de alumno del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno2.numero);
  textcolor(green); write('El promedio del alumno '); textcolor(yellow); write(alumno);
textcolor(green); write(' es '); textcolor(red); writeln(registro_alumno2.promedio:0:2);
end;
procedure imprimir_lista_alumnos2(lista_alumnos2: t_lista_alumnos2);
var
  i: int16;
begin
  i:=0;
  while (lista_alumnos2<>nil) do
  begin
    i:=i+1;
    textcolor(green); write('La información del alumno '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
    imprimir_registro_alumno2(lista_alumnos2^.ele,i);
    writeln();
    lista_alumnos2:=lista_alumnos2^.sig;
  end;
end;
var
  lista_alumnos1: t_lista_alumnos1;
  lista_alumnos2: t_lista_alumnos2;
begin
  randomize;
  lista_alumnos1:=nil;
```

```

lista_alumnos2:=nil;
writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
cargar_lista_alumnos1(lista_alumnos1);
imprimir_lista_alumnos1(lista_alumnos1);
writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
cargar_lista_alumnos2(lista_alumnos2,lista_alumnos1);
imprimir_lista_alumnos2(lista_alumnos2);
end.

```

(c) *Analizar: ¿Qué cambios requieren los incisos (a) y (b), si no se sabe de antemano la cantidad de materias aprobadas de cada alumno y si, además, se desean registrar los aplazos? ¿Cómo puede diseñarse una solución modularizada que requiera la menor cantidad de cambios?*

```

program TP0_E1c;
{$codepage UTF8}
uses crt;
const
  anio_ini=2000; anio_fin=2023;
  nota_ini=1; nota_fin=10;
  nota_corte=4; nota_salida=0;
  numero_salida=11111;
type
  t_anio=anio_ini..anio_fin;
  t_nota=nota_ini..nota_fin;
  t_lista_notas=^t_nodo_notas;
  t_nodo_notas=record
    ele: t_nota;
    sig: t_lista_notas;
  end;
  t_registro_alumno1=record
    apellido: string;
    numero: int32;
    anio_ingreso: t_anio;
    notas: t_lista_notas;
    examenes_rendidos: int16;
    materias_aprobadas: int8;
  end;
  t_registro_alumno2=record
    numero: int32;
    promedio_con_aplazos: real;
    promedio_sin_aplazos: real;
  end;
  t_lista_alumnos1=^t_nodo_alumnos1;
  t_nodo_alumnos1=record
    ele: t_registro_alumno1;
    sig: t_lista_alumnos1;
  end;
  t_lista_alumnos2=^t_nodo_alumnos2;
  t_nodo_alumnos2=record
    ele: t_registro_alumno2;
    sig: t_lista_alumnos2;
  end;
function random_string(length: int8): string;
var
  i: int8;
  string_aux: string;
begin
  string_aux:='';
  for i:= 1 to length do
    string_aux:=string_aux+chr(ord('A')+random(26));
  random_string:=string_aux;

```

```
end;
procedure agregar_adelante_lista_notas(var lista_notas: t_lista_notas; nota: t_nota);
var
    nuevo: t_lista_notas;
begin
    new(nuevo);
    nuevo^.ele:=nota;
    nuevo^.sig:=lista_notas;
    lista_notas:=nuevo;
end;
procedure leer_nota(var nota: int8);
var
    i: int8;
begin
    i:=random(100);
    if (i=0) then
        nota:=nota_salida
    else
        nota:=nota_ini+random(nota_fin);
    end;
end;
procedure leer_alumno(var registro_alumno1: t_registro_alumno1);
var
    nota: int8;
    materias_aprobadas, i: int8;
    examenes_rendidos: int16;
begin
    registro_alumno1.notas:=nil;
    examenes_rendidos:=0; materias_aprobadas:=0;
    registro_alumno1.apellido:=random_string(5+random(6));
    i:=random(100);
    if (i=0) then
        registro_alumno1.numero:=numero_salida
    else
        registro_alumno1.numero:=1+random(high(int16));
        registro_alumno1.anio_ingreso:=anio_ini+random(anio_fin-anio_ini+1);
        leer_nota(nota);
        while (nota<>nota_salida) do
            begin
                agregar_adelante_lista_notas(registro_alumno1.notas,nota);
                examenes_rendidos:=examenes_rendidos+1;
                if (nota>=nota_corte) then
                    materias_aprobadas:=materias_aprobadas+1;
                leer_nota(nota);
            end;
            registro_alumno1.examenes_rendidos:=examenes_rendidos;
            registro_alumno1.materias_aprobadas:=materias_aprobadas;
        end;
end;
procedure agregar_adelante_lista_alumnos1(var lista_alumnos1: t_lista_alumnos1;
registro_alumno1: t_registro_alumno1);
var
    nuevo: t_lista_alumnos1;
begin
    new(nuevo);
    nuevo^.ele:=registro_alumno1;
    nuevo^.sig:=lista_alumnos1;
    lista_alumnos1:=nuevo;
end;
procedure cargar_lista_alumnos1(var lista_alumnos1: t_lista_alumnos1);
var
    registro_alumno1: t_registro_alumno1;
begin
    repeat
        leer_alumno(registro_alumno1);
        agregar_adelante_lista_alumnos1(lista_alumnos1,registro_alumno1);
    until (registro_alumno1.numero=numero_salida);
end;
```

```

procedure imprimir_registro_alumno1(registro_alumno1: t_registro_alumno1; alumno: int16);
begin
    textcolor(green); write('El apellido del alumno '); textcolor(yellow); write(alumno);
textcolor(green); write(' es '); textcolor(red); writeln(registro_alumno1.apellido);
    textcolor(green); write('El número de alumno del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno1.numero);
    textcolor(green); write('El año de ingreso del alumno '); textcolor(yellow); write(alumno);
textcolor(green); write(' es '); textcolor(red); writeln(registro_alumno1.anio_ingreso);
    textcolor(green); write('La cantidad de exámenes rendidos del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno1.examenes_rendidos);
    textcolor(green); write('La cantidad de materias aprobadas del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno1.materias_aprobadas);
end;
procedure imprimir_lista_alumnos1(lista_alumnos1: t_lista_alumnos1);
var
    i: int16;
begin
    i:=0;
    while (lista_alumnos1<>nil) do
        begin
            i:=i+1;
            textcolor(green); write('La información del alumno '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
            imprimir_registro_alumno1(lista_alumnos1^.ele,i);
            writeln();
            lista_alumnos1:=lista_alumnos1^.sig;
        end;
    end;
end;
procedure cargar_registro_alumno2(var registro_alumno2: t_registro_alumno2; registro_alumno1:
t_registro_alumno1);
var
    suma_con_aplazos, suma_sin_aplazos: int16;
begin
    suma_con_aplazos:=0; suma_sin_aplazos:=0;
    registro_alumno2.numero:=registro_alumno1.numero;
    if (registro_alumno1.examenes_rendidos<>0) then
        begin
            while (registro_alumno1.notas<>nil) do
                begin
                    suma_con_aplazos:=suma_con_aplazos+registro_alumno1.notas^.ele;
                    if (registro_alumno1.notas^.ele>=nota_corte) then
                        suma_sin_aplazos:=suma_sin_aplazos+registro_alumno1.notas^.ele;
                    registro_alumno1.notas:=registro_alumno1.notas^.sig;
                end;
            registro_alumno2.promedio_con_aplazos:=suma_con_aplazos/registro_alumno1.examenes_rendidos
;
            if (registro_alumno1.materias_aprobadas<>0) then
                registro_alumno2.promedio_sin_aplazos:=suma_sin_aplazos/registro_alumno1.materias_aproba
das
            else
                registro_alumno2.promedio_sin_aplazos:=suma_sin_aplazos;
            end
        else
            begin
                registro_alumno2.promedio_con_aplazos:=suma_con_aplazos;
                registro_alumno2.promedio_sin_aplazos:=suma_sin_aplazos;
            end;
        end;
end;
procedure agregar_adelante_lista_alumnos2(var lista_alumnos2: t_lista_alumnos2;
registro_alumno2: t_registro_alumno2);
var
    nuevo: t_lista_alumnos2;
begin

```

```

    new(nuevo);
    nuevo^.ele:=registro_alumno2;
    nuevo^.sig:=lista_alumnos2;
    lista_alumnos2:=nuevo;
end;
procedure cargar_lista_alumnos2(var lista_alumnos2: t_lista_alumnos2; lista_alumnos1:
t_lista_alumnos1);
var
    registro_alumno2: t_registro_alumno2;
begin
    while (lista_alumnos1<>nil) do
        begin
            cargar_registro_alumno2(registro_alumno2,lista_alumnos1^.ele);
            agregar_adelante_lista_alumnos2(lista_alumnos2,registro_alumno2);
            lista_alumnos1:=lista_alumnos1^.sig;
        end;
    end;
end;
procedure imprimir_registro_alumno2(registro_alumno2: t_registro_alumno2; alumno: int16);
begin
    textcolor(green); write('El número de alumno del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno2.numero);
    textcolor(green); write('El promedio CON aplazos del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno2.promedio_con_aplazos:0:2);
    textcolor(green); write('El promedio SIN aplazos del alumno '); textcolor(yellow);
write(alumno); textcolor(green); write(' es '); textcolor(red);
writeln(registro_alumno2.promedio_sin_aplazos:0:2);
end;
procedure imprimir_lista_alumnos2(lista_alumnos2: t_lista_alumnos2);
var
    i: int16;
begin
    i:=0;
    while (lista_alumnos2<>nil) do
        begin
            i:=i+1;
            textcolor(green); write('La información del alumno '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
            imprimir_registro_alumno2(lista_alumnos2^.ele,i);
            writeln();
            lista_alumnos2:=lista_alumnos2^.sig;
        end;
    end;
end;
var
    lista_alumnos1: t_lista_alumnos1;
    lista_alumnos2: t_lista_alumnos2;
begin
    randomize;
    lista_alumnos1:=nil;
    lista_alumnos2:=nil;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_lista_alumnos1(lista_alumnos1);
    imprimir_lista_alumnos1(lista_alumnos1);
    writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
    cargar_lista_alumnos2(lista_alumnos2,lista_alumnos1);
    imprimir_lista_alumnos2(lista_alumnos2);
end.

```

Ejercicio 2.

Implementar un programa que procese información de propiedades que están a la venta en una inmobiliaria.

(a) Implementar un módulo para almacenar, en una estructura adecuada, las propiedades agrupadas por zona. Las propiedades de una misma zona deben quedar almacenadas ordenadas por tipo de propiedad. Para cada propiedad, debe almacenarse el código, el tipo de propiedad y el precio total. De cada propiedad, se lee: zona (1 a 5), código de propiedad, tipo de propiedad, cantidad de metros cuadrados y precio del metro cuadrado. La lectura finaliza cuando se ingresa el precio del metro cuadrado -1.

(b) Implementar un módulo que reciba la estructura generada en (a), un número de zona y un tipo de propiedad y retorne los códigos de las propiedades de la zona recibida y del tipo recibido.

```

program TP0_E2;
{$codepage UTF8}
uses crt;
const
  zona_ini=1; zona_fin=5;
  tipo_ini=1; tipo_fin=3;
  preciom2_salida=-1.0;
type
  t_zona=zona_ini..zona_fin;
  t_tipo=tipo_ini..tipo_fin;
  t_registro_propiedad1=record
    zona: t_zona;
    codigo: int16;
    tipo: t_tipo;
    m2: real;
    preciom2: real;
  end;
  t_registro_propiedad2=record
    codigo: int16;
    tipo: t_tipo;
    precio_total: real;
  end;
  t_lista_propiedades1=^t_nodo_propiedades1;
  t_nodo_propiedades1=record
    ele: t_registro_propiedad2;
    sig: t_lista_propiedades1;
  end;
  t_lista_propiedades2=^t_nodo_propiedades2;
  t_nodo_propiedades2=record
    ele: int16;
    sig: t_lista_propiedades2;
  end;
  t_vector_propiedades=array[t_zona] of t_lista_propiedades1;
procedure inicializar_vector_propiedades(var vector_propiedades: t_vector_propiedades);
var
  i: t_zona;
begin
  for i:= zona_ini to zona_fin do
    vector_propiedades[i]:=nil;
  end;
procedure leer_propiedad(var registro_propiedad1: t_registro_propiedad1);
var
  i: int8;
begin

```



```

i:=random(100);
if (i=0) then
    registro_propiedad1.preciom2:=preciom2_salida
else
    registro_propiedad1.preciom2:=1+random(100);
    if (registro_propiedad1.preciom2<>preciom2_salida) then
        begin
            registro_propiedad1.zona:=zona_ini+random(zona_fin);
            registro_propiedad1.codigo:=1+random(high(int16));
            registro_propiedad1.tipo:=tipo_ini+random(tipo_fin);
            registro_propiedad1.m2:=1+random(100);
        end;
    end;
end;
procedure cargar_registro_propiedad2(var registro_propiedad2: t_registro_propiedad2;
registro_propiedad1: t_registro_propiedad1);
begin
    registro_propiedad2.codigo:=registro_propiedad1.codigo;
    registro_propiedad2.tipo:=registro_propiedad1.tipo;
    registro_propiedad2.precio_total:=registro_propiedad1.m2*registro_propiedad1.preciom2;
end;
procedure agregar_ordenado_lista_propiedades1(var lista_propiedades1: t_lista_propiedades1;
registro_propiedad1: t_registro_propiedad1);
var
    anterior, actual, nuevo: t_lista_propiedades1;
begin
    new(nuevo);
    cargar_registro_propiedad2(nuevo^.ele,registro_propiedad1);
    actual:=lista_propiedades1;
    while ((actual<>nil) and (actual^.ele.tipo<nuevo^.ele.tipo)) do
        begin
            anterior:=actual;
            actual:=actual^.sig;
        end;
    if (actual=lista_propiedades1) then
        lista_propiedades1:=nuevo
    else
        anterior^.sig:=nuevo;
        nuevo^.sig:=actual;
    end;
end;
procedure cargar_vector_propiedades(var vector_propiedades: t_vector_propiedades);
var
    registro_propiedad1: t_registro_propiedad1;
begin
    leer_propiedad(registro_propiedad1);
    while (registro_propiedad1.preciom2<>preciom2_salida) do
        begin
            agregar_ordenado_lista_propiedades1(vector_propiedades[registro_propiedad1.zona],registro_
propiedad1);
            leer_propiedad(registro_propiedad1);
        end;
    end;
end;
procedure imprimir_registro_propiedad2(registro_propiedad2: t_registro_propiedad2; zona:
t_zona; propiedad: int16);
begin
    textcolor(green); write('El código de la propiedad '); textcolor(yellow); write(propiedad);
textcolor(green); write(' de la zona '); textcolor(yellow); write(zona); textcolor(green);
write(' es '); textcolor(red); writeln(registro_propiedad2.codigo);
    textcolor(green); write('El tipo de la propiedad '); textcolor(yellow); write(propiedad);
textcolor(green); write(' de la zona '); textcolor(yellow); write(zona); textcolor(green);
write(' es '); textcolor(red); writeln(registro_propiedad2.tipo);
    textcolor(green); write('El precio total de la propiedad '); textcolor(yellow);
write(propiedad); textcolor(green); write(' de la zona '); textcolor(yellow); write(zona);
textcolor(green); write(' es '); textcolor(red);
writeln(registro_propiedad2.precio_total:0:2);
end;
procedure imprimir_lista_propiedades1(lista_propiedades1: t_lista_propiedades1; zona: t_zona);

```

```

var
  i: int16;
begin
  i:=0;
  while (lista_propiedades1<>nil) do
    begin
      i:=i+1;
      textcolor(green); write('La información de la propiedad '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
      imprimir_registro_propiedad2(lista_propiedades1^.ele,zona,i);
      writeln();
      lista_propiedades1:=lista_propiedades1^.sig;
    end;
  end;
procedure imprimir_vector_propiedades(vector_propiedades: t_vector_propiedades);
var
  i: t_zona;
begin
  for i:= zona_ini to zona_fin do
    imprimir_lista_propiedades1(vector_propiedades[i],i);
  end;
procedure agregar_adelante_lista_propiedades2(var lista_propiedades2: t_lista_propiedades2;
codigo: int16);
var
  nuevo: t_lista_propiedades2;
begin
  new(nuevo);
  nuevo^.ele:=codigo;
  nuevo^.sig:=lista_propiedades2;
  lista_propiedades2:=nuevo;
end;
procedure cargar_lista_propiedades2(var lista_propiedades2: t_lista_propiedades2;
vector_propiedades: t_vector_propiedades; zona: t_zona; tipo: t_tipo);
begin
  while ((vector_propiedades[zona]<>nil) and (vector_propiedades[zona]^^.ele.tipo<=tipo)) do
    begin
      if (vector_propiedades[zona]^^.ele.tipo=tipo) then
        agregar_adelante_lista_propiedades2(lista_propiedades2,vector_propiedades[zona]^^.ele.codigo);
      vector_propiedades[zona]:=vector_propiedades[zona]^^.sig;
    end;
  end;
procedure imprimir_lista_propiedades2(lista_propiedades2: t_lista_propiedades2);
var
  i: int16;
begin
  i:=0;
  while (lista_propiedades2<>nil) do
    begin
      i:=i+1;
      textcolor(green); write('El código de la propiedad '); textcolor(yellow); write(i);
textcolor(green); write(' es '); textcolor(red); writeln(lista_propiedades2^.ele);
      lista_propiedades2:=lista_propiedades2^.sig;
    end;
  end;
var
  vector_propiedades: t_vector_propiedades;
  lista_propiedades2: t_lista_propiedades2;
  zona: t_zona;
  tipo: t_tipo;
begin
  randomize;
  inicializar_vector_propiedades(vector_propiedades);
  lista_propiedades2:=nil;
  writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
  cargar_vector_propiedades(vector_propiedades);

```

```
imprimir_vector_propiedades(vector_propiedades);  
writeln(); textcolor(red); writeln('INCISO (b):'); writeln();  
zona:=zona_ini+random(zona_fin);  
tipo:=tipo_ini+random(tipo_fin);  
cargar_lista_propiedades2(lista_propiedades2,vector_propiedades,zona,tipo);  
if (lista_propiedades2<>nil) then  
    imprimir_lista_propiedades2(lista_propiedades2);  
end.
```

Ejercicio 3.

Implementar un programa que procese las ventas de un supermercado. El supermercado dispone de una tabla con los precios y stocks de los 1000 productos que tiene a la venta.

(a) Implementar un módulo que retorne, en una estructura de datos adecuada, los tickets de las ventas. De cada venta, se lee código de venta y los productos vendidos. Las ventas finalizan con el código de venta -1. De cada producto, se lee código y cantidad de unidades solicitadas. Para cada venta, la lectura de los productos a vender finaliza con cantidad de unidades vendidas igual a 0. El ticket debe contener:

- Código de venta.
- Detalle (código de producto, cantidad y precio unitario) de los productos que se pudieron vender. En caso de no haber stock suficiente, se venderá la máxima cantidad posible.
- Monto total de la venta.

(b) Implementar un módulo que reciba la estructura generada en el inciso (a) y un código de producto y retorne la cantidad de unidades vendidas de ese código de producto.

```
program TP0_E3;
{$codepage UTF8}
uses crt;
const
  productos_total=1000;
  codigo_venta_salida=-1;
  cantidad_salida=0;
type
  t_producto=1..productos_total;
  t_registro_producto=record
    codigo_producto: int16;
    cantidad: int8;
    precio: real;
  end;
  t_lista_productos=^t_nodo_productos;
  t_nodo_productos=record
    ele: t_registro_producto;
    sig: t_lista_productos;
  end;
  t_registro_venta=record
    codigo_venta: int16;
    productos: t_lista_productos;
    monto_total: real;
  end;
  t_lista_ventas=^t_nodo_ventas;
  t_nodo_ventas=record
    ele: t_registro_venta;
    sig: t_lista_ventas;
  end;
  t_vector_productos=array[t_producto] of t_registro_producto;
procedure cargar_vector_productos(var vector_productos: t_vector_productos);
var
  i: t_producto;
begin
  for i:= 1 to productos_total do
    begin
      vector_productos[i].codigo_producto:=i;
      vector_productos[i].cantidad:=1+random(high(int8));
```

```

        vector_productos[i].precio:=1+random(100);
    end;
end;
function buscar_vector_productos(vector_productos: t_vector_productos; codigo_producto:
int16): t_producto;
var
    pos: t_producto;
begin
    pos:=1;
    while (vector_productos[pos].codigo_producto<>codigo_producto) do
        pos:=pos+1;
        buscar_vector_productos:=pos;
    end;
procedure actualizar_vector_productos(var vector_productos: t_vector_productos; var
registro_producto: t_registro_producto; pos: t_producto);
begin
    if (registro_producto.cantidad<vector_productos[pos].cantidad) then
        vector_productos[pos].cantidad:=vector_productos[pos].cantidad-registro_producto.cantidad
    else
        begin
            registro_producto.cantidad:=vector_productos[pos].cantidad;
            vector_productos[pos].cantidad:=0;
        end;
    end;
end;
procedure leer_producto(var registro_producto: t_registro_producto; var vector_productos:
t_vector_productos; var monto_total: real);
var
    pos: t_producto;
    i: int8;
begin
    i:=random(10);
    if (i=0) then
        registro_producto.cantidad:=cantidad_salida
    else
        registro_producto.cantidad:=1+random(high(int8));
    if (registro_producto.cantidad<>cantidad_salida) then
        begin
            registro_producto.codigo_producto:=1+random(productos_total);
            pos:=buscar_vector_productos(vector_productos,registro_producto.codigo_producto);
            actualizar_vector_productos(vector_productos,registro_producto,pos);
            registro_producto.precio:=vector_productos[pos].precio;
            monto_total:=monto_total+registro_producto.precio*registro_producto.cantidad;
        end;
    end;
end;
procedure agregar_adelante_lista_productos(var lista_productos: t_lista_productos;
registro_producto: t_registro_producto);
var
    nuevo: t_lista_productos;
begin
    new(nuevo);
    nuevo^.ele:=registro_producto;
    nuevo^.sig:=lista_productos;
    lista_productos:=nuevo;
end;
procedure cargar_lista_productos(var lista_productos: t_lista_productos; var vector_productos:
t_vector_productos; var monto_total: real);
var
    registro_producto: t_registro_producto;
begin
    leer_producto(registro_producto,vector_productos,monto_total);
    while (registro_producto.cantidad<>cantidad_salida) do
        begin
            agregar_adelante_lista_productos(lista_productos,registro_producto);
            leer_producto(registro_producto,vector_productos,monto_total);
        end;
    end;
end;

```

```

procedure leer_venta(var registro_venta: t_registro_venta; var vector_productos:
t_vector_productos);
var
  i: int8;
  monto_total: real;
begin
  i:=random(100);
  if (i=0) then
    registro_venta.codigo_venta:=codigo_venta_salida
  else
    registro_venta.codigo_venta:=1+random(high(int16));
  if (registro_venta.codigo_venta<>codigo_venta_salida) then
    begin
      registro_venta.productos:=nil; monto_total:=0;
      cargar_lista_productos(registro_venta.productos,vector_productos,monto_total);
      registro_venta.monto_total:=monto_total;
    end;
  end;
end;
procedure agregar_adelante_lista_ventas(var lista_ventas: t_lista_ventas; registro_venta:
t_registro_venta);
var
  nuevo: t_lista_ventas;
begin
  new(nuevo);
  nuevo^.ele:=registro_venta;
  nuevo^.sig:=lista_ventas;
  lista_ventas:=nuevo;
end;
procedure cargar_lista_ventas(var lista_ventas: t_lista_ventas; vector_productos:
t_vector_productos);
var
  registro_venta: t_registro_venta;
begin
  leer_venta(registro_venta,vector_productos);
  while (registro_venta.codigo_venta<>codigo_venta_salida) do
    begin
      agregar_adelante_lista_ventas(lista_ventas,registro_venta);
      leer_venta(registro_venta,vector_productos);
    end;
  end;
end;
procedure imprimir_registro_producto(registro_producto: t_registro_producto; venta: int16;
codigo: int16);
begin
  textcolor(green); write('El código de producto del producto '); textcolor(yellow);
write(codigo); textcolor(green); write(' de la venta '); textcolor(yellow); write(venta);
textcolor(green); write(' es '); textcolor(red); writeln(registro_producto.codigo_producto);
  textcolor(green); write('La cantidad del producto '); textcolor(yellow); write(codigo);
textcolor(green); write(' de la venta '); textcolor(yellow); write(venta); textcolor(green);
write(' es '); textcolor(red); writeln(registro_producto.cantidad);
  textcolor(green); write('El precio unitario del producto '); textcolor(yellow);
write(codigo); textcolor(green); write(' de la venta '); textcolor(yellow); write(venta);
textcolor(green); write(' es '); textcolor(red); writeln(registro_producto.precio:0:2);
end;
procedure imprimir_lista_productos(lista_productos: t_lista_productos; venta: int16);
var
  i: int16;
begin
  i:=0;
  while (lista_productos<>nil) do
    begin
      i:=i+1;
      imprimir_registro_producto(lista_productos^.ele,venta,i);
      lista_productos:=lista_productos^.sig;
    end;
  end;
end;
procedure imprimir_registro_venta(registro_venta: t_registro_venta; venta: int16);

```

```

begin
    textcolor(green); write('El código de venta de la venta '); textcolor(yellow); write(venta);
textcolor(green); write(' es '); textcolor(red); writeln(registro_venta.codigo_venta);
    textcolor(green); write('El monto total de la venta '); textcolor(yellow); write(venta);
textcolor(green); write(' es '); textcolor(red); writeln(registro_venta.monto_total:0:2);
    imprimir_lista_productos(registro_venta.productos,venta);
end;
procedure imprimir_lista_ventas(lista_ventas: t_lista_ventas);
var
    i: int16;
begin
    i:=0;
    while (lista_ventas<>nil) do
        begin
            i:=i+1;
            textcolor(green); write('La información de la venta '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
            imprimir_registro_venta(lista_ventas^.ele,i);
            writeln();
            lista_ventas:=lista_ventas^.sig;
        end;
    end;
end;
procedure buscar_lista_productos(lista_productos: t_lista_productos; codigo_producto: int16;
var ventas: int32);
begin
    while (lista_productos<>nil) do
        begin
            if (lista_productos^.ele.codigo_producto=codigo_producto) then
                ventas:=ventas+lista_productos^.ele.cantidad;
                lista_productos:=lista_productos^.sig;
            end;
        end;
    end;
end;
function buscar_lista_ventas(lista_ventas: t_lista_ventas; codigo_producto: int16): int32;
var
    ventas: int32;
begin
    ventas:=0;
    while (lista_ventas<>nil) do
        begin
            buscar_lista_productos(lista_ventas^.ele.productos,codigo_producto,ventas);
            lista_ventas:=lista_ventas^.sig;
        end;
    buscar_lista_ventas:=ventas;
end;
var
    vector_productos: t_vector_productos;
    lista_ventas: t_lista_ventas;
    codigo_producto: int16;
begin
    randomize;
    lista_ventas:=nil;
    cargar_vector_productos(vector_productos);
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_lista_ventas(lista_ventas,vector_productos);
    if (lista_ventas<>nil) then
        begin
            imprimir_lista_ventas(lista_ventas);
            writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
            codigo_producto:=1+random(productos_total);
            textcolor(green); write('La cantidad de unidades vendidas en la lista del código de
producto '); textcolor(yellow); write(codigo_producto); textcolor(green); write(' es ');
textcolor(red); write(buscar_lista_ventas(lista_ventas,codigo_producto));
        end;
    end.

```

Trabajo Práctico N° 1: **Módulo Imperativo (Ordenación).**

Ejercicio 1.

Se desea procesar la información de las ventas de productos de un comercio (como máximo, 50). Implementar un programa que invoque los siguientes módulos:

(a) Un módulo que retorne la información de las ventas en un vector. De cada venta, se conoce el día de la venta, código del producto (entre 1 y 15) y cantidad vendida (como máximo, 99 unidades). El código debe generarse automáticamente (random) y la cantidad se debe leer. El ingreso de las ventas finaliza con el día de venta 0 (no se procesa).

(b) Un módulo que muestre el contenido del vector resultante del inciso (a).

(c) Un módulo que ordene el vector de ventas por código.

(d) Un módulo que muestre el contenido del vector resultante del inciso (c).

(e) Un módulo que elimine, del vector ordenado, las ventas con código de producto entre dos valores que se ingresan como parámetros.

(f) Un módulo que muestre el contenido del vector resultante del inciso (e).

(g) Un módulo que retorne la información (ordenada por código de producto de menor a mayor) de cada código par de producto junto a la cantidad total de productos vendidos.

(h) Un módulo que muestre la información obtenida en el inciso (g).

```
program TP1_E1;
{$codepage UTF8}
uses crt;
const
    ventas_total=50;
    dia_ini=1; dia_fin=31;
    codigo_ini=1; codigo_fin=15;
    cantidad_total=99;
    dia_salida=0;
type
    t_venta=1..ventas_total;
    t_codigo=codigo_ini..codigo_fin;
    t_cantidad=1..cantidad_total;
    t_registro_venta=record
        dia: int8;
        codigo: t_codigo;
        cantidad: t_cantidad;
    end;
    t_vector_ventas=array[t_venta] of t_registro_venta;
    t_vector_cantidades=array[t_codigo] of int16;
procedure leer_venta(var registro_venta: t_registro_venta);
var
    i: int8;
begin
```



```
i:=random(100);
if (i=0) then
    registro_venta.dia:=dia_salida
else
    registro_venta.dia:=dia_ini+random(dia_fin);
if (registro_venta.dia<>dia_salida) then
begin
    registro_venta.codigo:=codigo_ini+random(codigo_fin);
    registro_venta.cantidad:=1+random(cantidad_total);
end;
end;
procedure cargar_vector_ventas(var vector_ventas: t_vector_ventas; var ventas: int8);
var
    registro_venta: t_registro_venta;
begin
    leer_venta(registro_venta);
    while ((registro_venta.dia<>dia_salida) and (ventas<ventas_total)) do
    begin
        ventas:=ventas+1;
        vector_ventas[ventas]:=registro_venta;
        leer_venta(registro_venta);
    end;
end;
procedure imprimir_registro_venta(registro_venta: t_registro_venta; venta: t_venta);
begin
    textcolor(green); write('El día de la venta '); textcolor(yellow); write(venta);
textcolor(green); write(' es '); textcolor(red); writeln(registro_venta.dia);
    textcolor(green); write('El código de producto de la venta'); textcolor(yellow);
write(venta); textcolor(green); write(' es '); textcolor(red); writeln(registro_venta.codigo);
    textcolor(green); write('La cantidad vendida del producto de la venta '); textcolor(yellow);
write(venta); textcolor(green); write(' es '); textcolor(red);
writeln(registro_venta.cantidad);
end;
procedure imprimir_vector_ventas(vector_ventas: t_vector_ventas; ventas: int8);
var
    i: t_venta;
begin
    for i:= 1 to ventas do
    begin
        textcolor(green); write('La información de la venta '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
        imprimir_registro_venta(vector_ventas[i],i);
        writeln();
    end;
end;
procedure ordenar_vector_ventas(var vector_ventas: t_vector_ventas; ventas: int8);
var
    item: t_registro_venta;
    i, j, k: t_venta;
begin
    for i:= 1 to (ventas-1) do
    begin
        k:=i;
        for j:= (i+1) to ventas do
            if (vector_ventas[j].codigo<vector_ventas[k].codigo) then
                k:=j;
            item:=vector_ventas[k];
            vector_ventas[k]:=vector_ventas[i];
            vector_ventas[i]:=item;
        end;
    end;
end;
procedure verificar_codigos(var codigo1, codigo2: t_codigo);
var
    aux: t_codigo;
begin
    if (codigo1>codigo2) then
```

```

begin
    aux:=codigo1;
    codigo1:=codigo2;
    codigo2:=aux;
end;
end;
procedure eliminar_vector_ventas(var vector_ventas: t_vector_ventas; var ventas: int8;
codigo1, codigo2: t_codigo);
var
    i, i_izq, i_der, salto: t_codigo;
begin
    i:=1;
    while ((i<ventas) and (vector_ventas[i].codigo<=codigo1)) do
        i:=i+1;
    end;
    i_izq:=i;
    while ((i<ventas) and (vector_ventas[i].codigo<codigo2)) do
        i:=i+1;
    end;
    i_der:=i;
    salto:=i_der-i_izq;
    while (i_izq+salto<=ventas) do
        begin
            vector_ventas[i_izq]:=vector_ventas[i_izq+salto];
            i_izq:=i_izq+1;
        end;
        ventas:=ventas-salto;
    end;
end;
procedure inicializar_vector_cantidades(var vector_cantidades: t_vector_cantidades);
var
    i: t_codigo;
begin
    for i:= codigo_ini to codigo_fin do
        begin
            vector_cantidades[i]:=0;
        end;
    end;
end;
procedure cargar_vector_cantidades(var vector_cantidades: t_vector_cantidades; vector_ventas:
t_vector_ventas; ventas: int8);
var
    i: t_venta;
    codigo: t_codigo;
begin
    for i:= 1 to ventas do
        begin
            codigo:=vector_ventas[i].codigo;
            if (codigo mod 2=0) then
                vector_cantidades[codigo]:=vector_cantidades[codigo]+vector_ventas[i].cantidad;
            end;
        end;
    end;
end;
procedure imprimir_vector_cantidades(vector_cantidades: t_vector_cantidades);
var
    i: t_codigo;
begin
    for i:= codigo_ini to codigo_fin do
        begin
            textcolor(green); write('La cantidad total de productos vendidos del código de producto
'); textcolor(yellow); write(i); textcolor(green); write(' es '); textcolor(red);
writeln(vector_cantidades[i]);
        end;
    end;
end;
var
    vector_ventas: t_vector_ventas;
    vector_cantidades: t_vector_cantidades;
    codigo1, codigo2: t_codigo;
    ventas: int8;
begin
    randomize;

```

```
ventas:=0;
inicializar_vector_cantidades(vector_cantidades);
writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
cargar_vector_ventas(vector_ventas,ventas);
if (ventas<>0) then
begin
  writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
  imprimir_vector_ventas(vector_ventas,ventas);
  writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
  ordenar_vector_ventas(vector_ventas,ventas);
  writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
  imprimir_vector_ventas(vector_ventas,ventas);
  writeln(); textcolor(red); writeln('INCISO (e):'); writeln();
  codigo1:=codigo_ini+random(codigo_fin); codigo2:=codigo_ini+random(codigo_fin);
  verificar_codigos(codigo1,codigo2);
  eliminar_vector_ventas(vector_ventas,ventas,codigo1,codigo2);
  if (ventas<>0) then
  begin
    writeln(); textcolor(red); writeln('INCISO (f):'); writeln();
    imprimir_vector_ventas(vector_ventas,ventas);
    writeln(); textcolor(red); writeln('INCISO (g):'); writeln();
    cargar_vector_cantidades(vector_cantidades,vector_ventas,ventas);
    writeln(); textcolor(red); writeln('INCISO (h):'); writeln();
    imprimir_vector_cantidades(vector_cantidades);
  end;
end;
end.
```

Ejercicio 2.

El administrador de un edificio de oficinas cuenta, en papel, con la información del pago de las expensas de dichas oficinas. Implementar un programa que invoque a módulos para cada uno de los siguientes puntos:

(a) *Generar un vector, sin orden, con, a lo sumo, las 300 oficinas que administra. De cada oficina, se ingresa el código de identificación, DNI del propietario y valor de la expensa. La lectura finaliza cuando se ingresa el código de identificación -1, el cual no se procesa.*

(b) *Ordenar el vector, aplicando el método de inserción, por código de identificación de la oficina.*

(c) *Ordenar el vector aplicando el método de selección, por código de identificación de la oficina.*

```
program TP1_E2;
{$codepage UTF8}
uses crt;
const
  oficinas_total=300;
  codigo_salida=-1;
type
  t_oficina=1..oficinas_total;
  t_registro_oficina=record
    codigo: int16;
    dni: int32;
    expensa: real;
  end;
  t_vector_oficinas=array[t_oficina] of t_registro_oficina;
procedure leer_oficina(var registro_oficina: t_registro_oficina);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_oficina.codigo:=codigo_salida
  else
    registro_oficina.codigo:=1+random(high(int16));
    if (registro_oficina.codigo<>codigo_salida) then
      begin
        registro_oficina.dni:=1+random(high(int32));
        registro_oficina.expensa:=1+random(100);
      end;
    end;
end;
procedure cargar_vector_oficinas(var vector_oficinas: t_vector_oficinas; var oficinas: int16);
var
  registro_oficina: t_registro_oficina;
begin
  leer_oficina(registro_oficina);
  while (registro_oficina.codigo<>codigo_salida) and (oficinas<oficinas_total) do
    begin
      oficinas:=oficinas+1;
      vector_oficinas[oficinas]:=registro_oficina;
      leer_oficina(registro_oficina);
    end;
  end;
end;
procedure imprimir_registro_oficina(registro_oficina: t_registro_oficina; oficina: t_oficina);
```

```

begin
    textcolor(green); write('El código de identificación de la oficina '); textcolor(yellow);
write(oficina); textcolor(green); write(' es '); textcolor(red);
writeln(registro_oficina.codigo);
    textcolor(green); write('El DNI del propietario de la oficina '); textcolor(yellow);
write(oficina); textcolor(green); write(' es '); textcolor(red);
writeln(registro_oficina.dni);
    textcolor(green); write('El valor de la expensa de la oficina '); textcolor(yellow);
write(oficina); textcolor(green); write(' es '); textcolor(red);
writeln(registro_oficina.expensa:0:2);
end;
procedure imprimir_vector_oficinas(vector_oficinas: t_vector_oficinas; oficinas: int16);
var
    i: t_oficina;
begin
    for i:= 1 to oficinas do
        begin
            textcolor(green); write('La información de la oficina '); textcolor(yellow); write(i);
textcolor(green); writeln(' es:');
            imprimir_registro_oficina(vector_oficinas[i],i);
            writeln();
        end;
    end;
procedure ordenacion_insercion_vector_oficinas(var vector_oficinas: t_vector_oficinas;
oficinas: int16);
var
    actual: t_registro_oficina;
    i, j: t_oficina;
begin
    for i:= 2 to oficinas do
        begin
            actual:=vector_oficinas[i];
            j:=i-1;
            while ((j>0) and (vector_oficinas[j].codigo>actual.codigo)) do
                begin
                    vector_oficinas[j+1]:=vector_oficinas[j];
                    j:=j-1;
                end;
            vector_oficinas[j+1]:=actual;
        end;
    end;
procedure ordenacion_seleccion_vector_oficinas(var vector_oficinas: t_vector_oficinas;
oficinas: int16);
var
    item: t_registro_oficina;
    i, j, k: t_oficina;
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];
            vector_oficinas[k]:=vector_oficinas[i];
            vector_oficinas[i]:=item;
        end;
    end;
var
    vector_oficinas: t_vector_oficinas;
    oficinas: int16;
begin
    randomize;
    oficinas:=0;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_vector_oficinas(vector_oficinas,oficinas);

```

```
if (oficinas>0) then
begin
  imprimir_vector_oficinas(vector_oficinas,oficinas);
  writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
  ordenacion_insercion_vector_oficinas(vector_oficinas,oficinas);
  imprimir_vector_oficinas(vector_oficinas,oficinas);
  writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
  ordenacion_seleccion_vector_oficinas(vector_oficinas,oficinas);
  imprimir_vector_oficinas(vector_oficinas,oficinas);
end;
end.
```

Ejercicio 3.

Netflix ha publicado la lista de películas que estarán disponibles durante el mes de diciembre de 2022. De cada película, se conoce: código de película, código de género (1: acción, 2: aventura, 3: drama, 4: suspenso, 5: comedia, 6: bélico, 7: documental y 8: terror) y puntaje promedio otorgado por las críticas. Implementar un programa que invoque a módulos para cada uno de los siguientes puntos:

(a) Leer los datos de películas, almacenarlos por orden de llegada y agrupados por código de género y retorne en una estructura de datos adecuada. La lectura finaliza cuando se lee el código de la película -1.

(b) Generar y retornar, en un vector, para cada género, el código de película con mayor puntaje obtenido entre todas las críticas, a partir de la estructura generada en (a).

(c) Ordenar los elementos del vector generado en (b) por puntaje, utilizando alguno de los dos métodos vistos en la teoría.

(d) Mostrar el código de película con mayor puntaje y el código de película con menor puntaje, del vector obtenido en el inciso (c).

```
program TP1_E3;
{$codepage UTF8}
uses crt;
const
    genero_ini=1; genero_fin=8;
    codigo_salida=-1;
type
    t_genero=genero_ini..genero_fin;
    t_registro_pelicula1=record
        codigo: int16;
        genero: t_genero;
        puntaje: real;
    end;
    t_registro_pelicula2=record
        codigo: int16;
        puntaje: real;
    end;
    t_lista_peliculas=^t_nodo_peliculas;
    t_nodo_peliculas=record
        ele: t_registro_pelicula2;
        sig: t_lista_peliculas;
    end;
    t_vector_peliculas1=array[t_genero] of t_lista_peliculas;
    t_vector_peliculas2=array[t_genero] of t_registro_pelicula2;
procedure inicializar_vector_peliculas1(var vector_peliculas1: t_vector_peliculas1);
var
    i: t_genero;
begin
    for i:= genero_ini to genero_fin do
        vector_peliculas1[i]:=nil;
    end;
procedure leer_pelicula(var registro_pelicula1: t_registro_pelicula1);
var
    i: int8;
begin
    i:=random(100);
    if (i=0) then
```

```

    registro_pelicula1.codigo:=codigo_salida
else
    registro_pelicula1.codigo:=1+random(high(int16));
if (registro_pelicula1.codigo<>codigo_salida) then
begin
    registro_pelicula1.genero:=genero_ini+random(genero_fin);
    registro_pelicula1.puntaje:=1+random(10);
end;
end;
procedure cargar_registro_pelicula2(var registro_pelicula2: t_registro_pelicula2;
registro_pelicula1: t_registro_pelicula1);
begin
    registro_pelicula2.codigo:=registro_pelicula1.codigo;
    registro_pelicula2.puntaje:=registro_pelicula1.puntaje;
end;
procedure agregar_atras_lista_peliculas(var lista_peliculas: t_lista_peliculas;
registro_pelicula1: t_registro_pelicula1);
var
    aux, ult: t_lista_peliculas;
begin
    new(aux);
    cargar_registro_pelicula2(aux^.ele,registro_pelicula1);
    aux^.sig:=nil;
    if (lista_peliculas=nil) then
        lista_peliculas:=aux
    else
        begin
            ult:=lista_peliculas;
            while (ult^.sig<>nil) do
                ult:=ult^.sig;
            end;
            ult^.sig:=aux;
        end;
    end;
end;
procedure cargar_vector_peliculas1(var vector_peliculas1: t_vector_peliculas1);
var
    registro_pelicula1: t_registro_pelicula1;
begin
    leer_pelicula(registro_pelicula1);
    while (registro_pelicula1.codigo<>codigo_salida) do
        begin
            agregar_atras_lista_peliculas(vector_peliculas1[registro_pelicula1.genero],registro_pelicula1);
            leer_pelicula(registro_pelicula1);
        end;
    end;
end;
procedure imprimir_registro_pelicula2(registro_pelicula2: t_registro_pelicula2; genero:
t_genero; pelicula: int16);
begin
    textcolor(green); write('El código de película de la película '); textcolor(yellow);
write(pelicula); textcolor(green); write(' del género '); textcolor(yellow); write(genero);
textcolor(green); write(' es '); textcolor(red); writeln(registro_pelicula2.codigo);
    textcolor(green); write('El puntaje de la película '); textcolor(yellow); write(pelicula);
textcolor(green); write(' del género '); textcolor(yellow); write(genero); textcolor(green);
write(' es '); textcolor(red); writeln(registro_pelicula2.puntaje:0:2);
end;
procedure imprimir_lista_peliculas(lista_peliculas: t_lista_peliculas; genero: t_genero);
var
    i: int16;
begin
    i:=0;
    while (lista_peliculas<>nil) do
        begin
            i:=i+1;
            imprimir_registro_pelicula2(lista_peliculas^.ele,genero,i);
            lista_peliculas:=lista_peliculas^.sig;
        end;
    end;
end;

```



```
end;
procedure imprimir_vector_peliculas1(vector_peliculas1: t_vector_peliculas1);
var
    i: t_genero;
begin
    for i:= genero_ini to genero_fin do
        begin
            textcolor(green); write('La información de las películas del género '); textcolor(yellow);
            write(i); textcolor(green); writeln(' es:');
            imprimir_lista_peliculas(vector_peliculas1[i],i);
            writeln();
        end;
    end;
end;
procedure cargar_vector_peliculas2(var vector_peliculas2: t_vector_peliculas2;
vector_peliculas1: t_vector_peliculas1);
var
    i: t_genero;
    codigo_max: int16;
    puntaje_max: real;
begin
    for i:= genero_ini to genero_fin do
        begin
            puntaje_max:=-9999999; codigo_max:=-1;
            while (vector_peliculas1[i]<>nil) do
                begin
                    if (vector_peliculas1[i]^ele.puntaje>puntaje_max) then
                        begin
                            puntaje_max:=vector_peliculas1[i]^ele.puntaje;
                            codigo_max:=vector_peliculas1[i]^ele.codigo;
                        end;
                    vector_peliculas1[i]:=vector_peliculas1[i]^sig;
                end;
            end;
            vector_peliculas2[i].codigo:=codigo_max;
            vector_peliculas2[i].puntaje:=puntaje_max;
        end;
    end;
end;
procedure imprimir_vector_peliculas2(vector_peliculas2: t_vector_peliculas2);
var
    i: t_genero;
begin
    for i:= genero_ini to genero_fin do
        begin
            imprimir_registro_pelicula2(vector_peliculas2[i],i,1);
            writeln();
        end;
    end;
end;
procedure ordenar_vector_peliculas2(var vector_peliculas2: t_vector_peliculas2);
var
    item: t_registro_pelicula2;
    i, j, k: t_genero;
begin
    for i:= genero_ini to (genero_fin-1) do
        begin
            k:=i;
            for j:= (i+1) to genero_fin do
                if (vector_peliculas2[j].puntaje<vector_peliculas2[k].puntaje) then
                    k:=j;
                item:=vector_peliculas2[k];
                vector_peliculas2[k]:=vector_peliculas2[i];
                vector_peliculas2[i]:=item;
            end;
        end;
    end;
end;
var
    vector_peliculas1: t_vector_peliculas1;
    vector_peliculas2: t_vector_peliculas2;
begin
```

```
randomize;
inicializar_vector_peliculas1(vector_peliculas1);
writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
cargar_vector_peliculas1(vector_peliculas1);
imprimir_vector_peliculas1(vector_peliculas1);
writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
cargar_vector_peliculas2(vector_peliculas2,vector_peliculas1);
imprimir_vector_peliculas2(vector_peliculas2);
writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
ordenar_vector_peliculas2(vector_peliculas2);
imprimir_vector_peliculas2(vector_peliculas2);
writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
textcolor(green); write('El código de película con mayor y menor puntaje son ');
textcolor(red); write(vector_peliculas2[genero_fin].codigo); textcolor(green); write(' y ');
textcolor(red); write(vector_peliculas2[genero_ini].codigo); textcolor(green); write(',
respectivamente');
end.
```

Ejercicio 4.

Una librería requiere el procesamiento de la información de sus productos. De cada producto, se conoce el código del producto, código de rubro (del 1 al 8) y precio. Implementar un programa que invoque a módulos para cada uno de los siguientes puntos:

(a) Leer los datos de los productos y almacenarlos ordenados por código de producto y agrupados por rubro, en una estructura de datos adecuada. El ingreso de los productos finaliza cuando se lee el precio 0.

(b) Una vez almacenados, mostrar los códigos de los productos pertenecientes a cada rubro.

(c) Generar un vector (de, a lo sumo, 30 elementos) con los productos del rubro 3. Considerar que puede haber más o menos de 30 productos del rubro 3. Si la cantidad de productos del rubro 3 es mayor a 30, almacenar los primeros 30 que están en la lista e ignorar el resto.

(d) Ordenar, por precio, los elementos del vector generado en (c) utilizando alguno de los dos métodos vistos en la teoría.

(e) Mostrar los precios del vector resultante del inciso (d).

(f) Calcular el promedio de los precios del vector resultante del inciso (d).

```
program TP1_E4;
{$codepage UTF8}
uses crt;
const
  rubro_ini=1; rubro_fin=8;
  precio_salida=0.0;
  productos_rubro3_total=30;
type
  t_rubro=rubro_ini..rubro_fin;
  t_registro_producto1=record
    codigo: int16;
    rubro: t_rubro;
    precio: real;
  end;
  t_registro_producto2=record
    codigo: int16;
    precio: real;
  end;
  t_lista_productos=^t_nodo_productos;
  t_nodo_productos=record
    ele: t_registro_producto2;
    sig: t_lista_productos;
  end;
  t_vector_productos1=array[t_rubro] of t_lista_productos;
  t_vector_productos2=array[1..productos_rubro3_total] of t_registro_producto2;
procedure inicializar_vector_productos1(var vector_productos1: t_vector_productos1);
var
  i: t_rubro;
begin
  for i:= rubro_ini to rubro_fin do
```

```

    vector_productos1[i]:=nil;
end;
procedure leer_producto(var registro_producto1: t_registro_producto1);
var
    i: int8;
begin
    i:=random(100);
    if (i=0) then
        registro_producto1.precio:=precio_salida
    else
        registro_producto1.precio:=1+random(100);
        if (registro_producto1.precio<>precio_salida) then
            begin
                registro_producto1.codigo:=1+random(high(int16));
                registro_producto1.rubro:=rubro_ini+random(rubro_fin);
            end;
        end;
end;
procedure cargar_registro_producto2(var registro_producto2: t_registro_producto2;
registro_producto1: t_registro_producto1);
begin
    registro_producto2.codigo:=registro_producto1.codigo;
    registro_producto2.precio:=registro_producto1.precio;
end;
procedure agregar_ordenado_lista_productos(var lista_productos: t_lista_productos;
registro_producto1: t_registro_producto1);
var
    anterior, actual, nuevo: t_lista_productos;
begin
    new(nuevo);
    cargar_registro_producto2(nuevo^.ele,registro_producto1);
    actual:=lista_productos;
    while ((actual<>nil) and (actual^.ele.codigo<nuevo^.ele.codigo)) do
        begin
            anterior:=actual;
            actual:=actual^.sig;
        end;
    if (actual=nil) then
        lista_productos:=nuevo
    else
        anterior^.sig:=nuevo;
        nuevo^.sig:=actual;
    end;
end;
procedure cargar_vector_productos1(var vector_productos1: t_vector_productos1);
var
    registro_producto1: t_registro_producto1;
begin
    leer_producto(registro_producto1);
    while (registro_producto1.precio<>precio_salida) do
        begin
            agregar_ordenado_lista_productos(vector_productos1[registro_producto1.rubro],registro_producto1);
            leer_producto(registro_producto1);
        end;
    end;
end;
procedure imprimir_registro_producto2(registro_producto2: t_registro_producto2; rubro:
t_rubro; producto: int16);
begin
    textcolor(green); write('El código de producto del producto '); textcolor(yellow);
write(producto); textcolor(green); write(' del código de rubro '); textcolor(yellow);
write(rubro); textcolor(green); write(' es '); textcolor(red);
writeln(registro_producto2.codigo);
    textcolor(green); write('El precio del producto '); textcolor(yellow); write(producto);
textcolor(green); write(' del código de rubro '); textcolor(yellow); write(rubro);
textcolor(green); write(' es '); textcolor(red); writeln(registro_producto2.precio:0:2);
end;
procedure imprimir_lista_productos(lista_productos: t_lista_productos; rubro: t_rubro);

```

```

var
  i: int16;
begin
  i:=0;
  while (lista_productos<>nil) do
    begin
      i:=i+1;
      imprimir_registro_producto2(lista_productos^.ele,rubro,i);
      lista_productos:=lista_productos^.sig;
    end;
  end;
procedure imprimir_vector_productos1(vector_productos1: t_vector_productos1);
var
  i: t_rubro;
begin
  for i:= rubro_ini to rubro_fin do
    begin
      textcolor(green); write('La información de los productos del rubro '); textcolor(yellow);
      write(i); textcolor(green); writeln(' es:');
      imprimir_lista_productos(vector_productos1[i],i);
      writeln();
    end;
  end;
procedure cargar_vector_productos2(var vector_productos2: t_vector_productos2; var
productos_rubro3: int8; lista_productos: t_lista_productos);
begin
  while ((lista_productos<>nil) and (productos_rubro3<productos_rubro3_total)) do
    begin
      productos_rubro3:=productos_rubro3+1;
      vector_productos2[productos_rubro3]:=lista_productos^.ele;
      lista_productos:=lista_productos^.sig;
    end;
  end;
procedure imprimir_vector_productos2(vector_productos2: t_vector_productos2; productos_rubro3:
int8);
var
  i: int8;
begin
  for i:= 1 to productos_rubro3 do
    begin
      textcolor(green); write('La información del producto '); textcolor(yellow); write(i);
      textcolor(green); writeln(' del rubro 3 son:');
      imprimir_registro_producto2(vector_productos2[i],3,i);
      writeln();
    end;
  end;
procedure ordenar_vector_productos2(var vector_productos2: t_vector_productos2;
productos_rubro3: int8);
var
  item: t_registro_producto2;
  i, j, k: int8;
begin
  for i:= 1 to (productos_rubro3-1) do
    begin
      k:=i;
      for j:= (i+1) to productos_rubro3 do
        if (vector_productos2[j].precio<vector_productos2[k].precio) then
          k:=j;
      item:=vector_productos2[k];
      vector_productos2[k]:=vector_productos2[i];
      vector_productos2[i]:=item;
    end;
  end;
function calcular_promedio_vector_productos2(vector_productos2: t_vector_productos2;
productos_rubro3: int8): real;
var

```

```
i: int8;
precio_total: real;
begin
    precio_total:=0;
    for i:= 1 to productos_rubro3 do
        precio_total:=precio_total+vector_productos2[i].precio;
        calcular_promedio_vector_productos2:=precio_total/productos_rubro3;
    end;
var
    vector_productos1: t_vector_productos1;
    vector_productos2: t_vector_productos2;
    productos_rubro3: int8;
begin
    randomize;
    productos_rubro3:=0;
    inicializar_vector_productos1(vector_productos1);
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_vector_productos1(vector_productos1);
    writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
    imprimir_vector_productos1(vector_productos1);
    writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
    cargar_vector_productos2(vector_productos2, productos_rubro3, vector_productos1[3]);
    if (productos_rubro3>0) then
    begin
        imprimir_vector_productos2(vector_productos2, productos_rubro3);
        writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
        ordenar_vector_productos2(vector_productos2, productos_rubro3);
        writeln(); textcolor(red); writeln('INCISO (e):'); writeln();
        imprimir_vector_productos2(vector_productos2, productos_rubro3);
        writeln(); textcolor(red); writeln('INCISO (f):'); writeln();
        textcolor(green); write('El promedio de los precios del vector_productos2 es ');
    end;
    textcolor(red);
    write(calcular_promedio_vector_productos2(vector_productos2, productos_rubro3):0:2);
    end;
end.
```

Trabajo Práctico N° 2: **Módulo Imperativo (Recursión).**

Ejercicio 1.

Implementar un programa que invoque a los siguientes módulos:

- (a) Un módulo recursivo que permita leer una secuencia de caracteres terminada en punto, los almacene en un vector con dimensión física igual a 10 y retorne el vector.*
- (b) Un módulo que reciba el vector generado en (a) e imprima el contenido del vector.*
- (c) Un módulo recursivo que reciba el vector generado en (a) e imprima el contenido del vector.*
- (d) Un módulo recursivo que permita leer una secuencia de caracteres terminada en punto y retorne la cantidad de caracteres leídos. El programa debe informar el valor retornado.*
- (e) Un módulo recursivo que permita leer una secuencia de caracteres terminada en punto y retorne una lista con los caracteres leídos.*
- (f) Un módulo recursivo que reciba la lista generada en (e) e imprima los valores de la lista en el mismo orden que están almacenados.*
- (g) Un módulo recursivo que reciba la lista generada en (e) e imprima los valores de la lista en orden inverso al que están almacenados.*

```
program TP2_E1;
{$codepage UTF8}
uses crt;
const
  char_salida='.';
  dimF=10;
type
  t_vector_chars=array[1..dimF] of char;
  t_lista_chars=^t_nodo_chars;
  t_nodo_chars=record
    ele: char;
    sig: t_lista_chars;
  end;
procedure leer_char(var c: char);
begin
  c:=chr(ord('.')+random(dimF));
end;
procedure cargar_vector_chars(var vector_chars: t_vector_chars; var dimL: int8);
var
  c: char;
begin
  leer_char(c);
  if ((dimL<dimF) and (c<>char_salida)) then
  begin
    dimL:=dimL+1;
    vector_chars[dimL]:=c;
    cargar_vector_chars(vector_chars,dimL);
  end;
```

```
end;
end;
procedure imprimir_secuencial_vector_chars(vector_chars: t_vector_chars; dimL: int8);
var
    i: int8;
begin
    for i:= 1 to dimL do
        begin
            textcolor(green); write('Elemento ',i,' del vector: '); textcolor(red);
            writeln(vector_chars[i]);
        end;
    end;
end;
procedure imprimir_recursoivo_vector_chars(vector_chars: t_vector_chars; dimL: int8);
begin
    if (dimL>0) then
        begin
            imprimir_recursoivo_vector_chars(vector_chars,dimL-1);
            textcolor(green); write('Elemento ',dimL,' del vector: '); textcolor(red);
            writeln(vector_chars[dimL]);
        end;
    end;
end;
function contar_chars(): int16;
var
    c: char;
begin
    leer_char(c);
    if (c=char_salida) then
        contar_chars:=0
    else
        contar_chars:=contar_chars()+1
    end;
end;
procedure agregar_adelante_lista_chars(var lista_chars: t_lista_chars; c: char);
var
    nuevo: t_lista_chars;
begin
    new(nuevo);
    nuevo^.ele:=c;
    nuevo^.sig:=lista_chars;
    lista_chars:=nuevo;
end;
procedure cargar_lista_chars(var lista_chars: t_lista_chars);
var
    c: char;
begin
    leer_char(c);
    if (c<>char_salida) then
        begin
            agregar_adelante_lista_chars(lista_chars,c);
            cargar_lista_chars(lista_chars);
        end;
    end;
end;
procedure imprimir1_lista_chars(lista_chars: t_lista_chars; i: int8);
begin
    if (lista_chars<>nil) then
        begin
            i:=i+1;
            textcolor(green); write('Elemento ',i,' de la lista: '); textcolor(red);
            writeln(lista_chars^.ele);
            imprimir1_lista_chars(lista_chars^.sig,i);
        end;
    end;
end;
procedure imprimir2_lista_chars(lista_chars: t_lista_chars; i: int8);
begin
    if (lista_chars<>nil) then
        begin
            i:=i+1;
```



```
    imprimir2_lista_chars(lista_chars^.sig,i);
    textcolor(green); write('Elemento ',i,' de la lista: '); textcolor(red);
writeln(lista_chars^.ele);
end;
end;
var
    vector_chars: t_vector_chars;
    lista_chars: t_lista_chars;
    dimL, i: int8;
begin
    randomize;
    dimL:=0;
    lista_chars:=nil;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_vector_chars(vector_chars,dimL);
    writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
    imprimir_secuencial_vector_chars(vector_chars,dimL);
    writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
    imprimir_recurso_vector_chars(vector_chars,dimL);
    writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
    textcolor(green); write('La cantidad de caracteres leídos es '); textcolor(red);
writeln(contar_chars());
    writeln(); textcolor(red); writeln('INCISO (e):'); writeln();
    cargar_lista_chars(lista_chars);
    if (lista_chars<>nil) then
    begin
        writeln(); textcolor(red); writeln('INCISO (f):'); writeln();
        i:=0;
        imprimir1_lista_chars(lista_chars,i);
        writeln(); textcolor(red); writeln('INCISO (g):'); writeln();
        i:=0;
        imprimir2_lista_chars(lista_chars,i);
    end;
end.
```

Ejercicio 2.

Realizar un programa que lea números hasta leer el valor 0 e imprima, para cada número leído, sus dígitos en el orden en que aparecen en el número. Debe implementarse un módulo recursivo que reciba el número e imprima lo pedido. Ejemplo, si se lee el valor 256, se debe imprimir 2 5 6.

```
program TP2_E2;
{$codepage UTF8}
uses crt;
const
  num_salida=0;
procedure leer_numero(var num: int8);
begin
  num:=num_salida+random(high(int8));
end;
procedure descomponer_numero(var digito: int8; var num: int16);
begin
  digito:=num mod 10;
  num:=num div 10;
end;
procedure imprimir_digitos(num: int16);
var
  digito: int8;
begin
  if (num<>num_salida) then
  begin
    descomponer_numero(digito,num);
    imprimir_digitos(num);
    textcolor(red); write(digito, ' ');
  end;
end;
procedure leer_numeros();
var
  num: int8;
begin
  leer_numero(num);
  if (num<>num_salida) then
  begin
    textcolor(green); writeln(); write('Número entero: '); textcolor(red); writeln(num);
    textcolor(green); write('Número entero (dígito por dígito): ');
    imprimir_digitos(num);
    writeln();
    leer_numeros();
  end;
end;
begin
  leer_numeros();
end.
```

Ejercicio 3.

Escribir un programa que:

(a) *Implemente un módulo recursivo que genere una lista de números enteros “random” mayores a 0 y menores a 100. Finalizar con el número 0.*

(b) *Implemente un módulo recursivo que devuelva el mínimo valor de la lista.*

(c) *Implemente un módulo recursivo que devuelva el máximo valor de la lista.*

(d) *Implemente un módulo recursivo que devuelva verdadero si un valor determinado se encuentra en la lista o falso en caso contrario.*

```
program TP2_E3;
{$codepage UTF8}
uses crt;
const
  num_ini=0; num_fin=100;
  num_salida=0;
type
  t_numero=num_ini..num_fin;
  t_lista_numeros=^t_nodo_numeros;
  t_nodo_numeros=record
    ele: int16;
    sig: t_lista_numeros;
  end;
procedure leer_numero(var num: t_numero);
begin
  num:=num_salida+random(num_fin);
end;
procedure agregar_adelante_lista_numeros(var lista_numeros: t_lista_numeros; num: t_numero);
var
  nuevo: t_lista_numeros;
begin
  new(nuevo);
  nuevo^.ele:=num;
  nuevo^.sig:=lista_numeros;
  lista_numeros:=nuevo;
end;
procedure cargar_lista_numeros(var lista_numeros: t_lista_numeros);
var
  num: t_numero;
begin
  leer_numero(num);
  if (num<>num_salida) then
  begin
    agregar_adelante_lista_numeros(lista_numeros,num);
    cargar_lista_numeros(lista_numeros);
  end;
end;
procedure imprimir_lista_numeros(lista_numeros: t_lista_numeros; i: int16);
begin
  if (lista_numeros<>nil) then
  begin
    i:=i+1;
    textcolor(green); write('Elemento ',i,' de la lista: '); textcolor(red);
    writeln(lista_numeros^.ele);
    imprimir_lista_numeros(lista_numeros^.sig,i);
  end;
end;
```

```
end;
procedure calcular_minimo_lista_numeros(lista_numeros: t_lista_numeros; var num_min:
t_numero);
begin
    if (lista_numeros<>nil) then
        begin
            if (lista_numeros^.ele<num_min) then
                num_min:=lista_numeros^.ele;
            calcular_minimo_lista_numeros(lista_numeros^.sig,num_min);
        end;
    end;
end;
procedure calcular_maximo_lista_numeros(lista_numeros: t_lista_numeros; var num_max:
t_numero);
begin
    if (lista_numeros<>nil) then
        begin
            if (lista_numeros^.ele>num_max) then
                num_max:=lista_numeros^.ele;
            calcular_maximo_lista_numeros(lista_numeros^.sig,num_max);
        end;
    end;
end;
function buscar_lista_numeros(lista_numeros: t_lista_numeros; num: int16): boolean;
begin
    if (lista_numeros=nil) then
        buscar_lista_numeros:=false
    else
        if (lista_numeros^.ele=num) then
            buscar_lista_numeros:=true
        else
            buscar_lista_numeros:=buscar_lista_numeros(lista_numeros^.sig,num);
        end;
    end;
end;
var
    lista_numeros: t_lista_numeros;
    num_min, num_max: t_numero;
    i, num: int16;
begin
    randomize;
    lista_numeros:=nil;
    num_min:=high(t_numero);
    num_max:=low(t_numero);
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_lista_numeros(lista_numeros);
    if (lista_numeros<>nil) then
        begin
            i:=0;
            imprimir_lista_numeros(lista_numeros,i);
            writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
            calcular_minimo_lista_numeros(lista_numeros,num_min);
            textcolor(green); write('El mínimo valor de la lista es '); textcolor(red);
            writeln(num_min);
            writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
            calcular_maximo_lista_numeros(lista_numeros,num_max);
            textcolor(green); write('El máximo valor de la lista es '); textcolor(red);
            writeln(num_max);
            writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
            num:=(num_ini+1)+random(num_fin-(num_ini+1));
            textcolor(green); write('¿El número '); textcolor(yellow); write(num); textcolor(green);
            write(' se encuentra en la lista?: '); textcolor(red);
            write(buscar_lista_numeros(lista_numeros,num));
        end;
    end;
end.
```

Ejercicio 4.

Escribir un programa con:

- (a) *Un módulo recursivo que retorne un vector de 20 números enteros “random” mayores a 0 y menores a 100.*
- (b) *Un módulo recursivo que devuelva el máximo valor del vector.*
- (c) *Un módulo recursivo que devuelva la suma de los valores contenidos en el vector.*

```

program TP2_E4;
{$codepage UTF8}
uses crt;
const
    dimF=20;
    num_ini=0; num_fin=100;
type
    t_numero=num_ini..num_fin;
    t_vector_numeros=array[1..dimF] of t_numero;
procedure cargar_vector_numeros(var vector_numeros: t_vector_numeros; var dimL: int8);
begin
    if (dimL<dimF) then
    begin
        dimL:=dimL+1;
        vector_numeros[dimL]:=(num_ini+1)+random(num_fin-(num_ini+1));
        cargar_vector_numeros(vector_numeros,dimL);
    end;
end;
procedure imprimir_vector_numeros(vector_numeros: t_vector_numeros; dimL: int8);
begin
    if (dimL>0) then
    begin
        imprimir_vector_numeros(vector_numeros,dimL-1);
        textcolor(green); write('Elemento ',dimL,' del vector: '); textcolor(red);
        writeln(vector_numeros[dimL]);
    end;
end;
procedure calcular_maximo_vector_numeros(vector_numeros: t_vector_numeros; dimL: int8; var
num_max: t_numero);
begin
    if (dimL>0) then
    begin
        if (vector_numeros[dimL]>num_max) then
            num_max:=vector_numeros[dimL];
        calcular_maximo_vector_numeros(vector_numeros,dimL-1,num_max);
    end;
end;
function sumar_vector_numeros(vector_numeros: t_vector_numeros; dimL: int8): int16;
begin
    if (dimL=1) then
        sumar_vector_numeros:=vector_numeros[dimL]
    else
        sumar_vector_numeros:=sumar_vector_numeros(vector_numeros,dimL-1)+vector_numeros[dimL];
end;
var
    vector_numeros: t_vector_numeros;
    num_max: t_numero;
    dimL: int8;
begin
    randomize;

```

```
dimL:=0;
num_max:=low(t_numero);
writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
cargar_vector_numeros(vector_numeros,dimL);
if (dimL>0) then
begin
  imprimir_vector_numeros(vector_numeros,dimL);
  writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
  calcular_maximo_vector_numeros(vector_numeros,dimL,num_max);
  textcolor(green); write('El máximo valor del vector es '); textcolor(red);
writeln(num_max);
  writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
  textcolor(green); write('La suma de los valores contenidos en el vector es ');
textcolor(red); write(sumar_vector_numeros(vector_numeros,dimL));
end;
end.
```

Ejercicio 5.

Implementar un módulo que realice una búsqueda dicotómica en un vector, utilizando el siguiente encabezado:

Procedure busquedaDicotomica(v: vector; ini,fin: indice; dato: integer; var pos: indice);

Nota: El parámetro “pos” debe retornar la posición del dato o -1 si el dato no se encuentra en el vector.

```

program TP2_E5;
{$codepage UTF8}
uses crt;
const
    dimF=10;
    num_salida=0;
type
    t_vector_numeros=array[1..dimF] of int8;
procedure cargar_vector_numeros(var vector_numeros: t_vector_numeros; var dimL: int8);
var
    num: int8;
begin
    if (dimL<dimF) then
    begin
        num:=num_salida+random(high(int8));
        if (num<>num_salida) then
        begin
            dimL:=dimL+1;
            vector_numeros[dimL]:=num;
            cargar_vector_numeros(vector_numeros,dimL);
        end;
    end;
end;
procedure imprimir_vector_numeros(vector_numeros: t_vector_numeros; dimL: int8);
begin
    if (dimL>0) then
    begin
        imprimir_vector_numeros(vector_numeros,dimL-1);
        textcolor(green); write('Elemento ',dimL,' del vector: '); textcolor(red);
        writeln(vector_numeros[dimL]);
    end;
end;
procedure ordenar_vector_numeros(var vector_numeros: t_vector_numeros; dimL: int8);
var
    i, j, k, item: int8;
begin
    for i:= 1 to (dimL-1) do
    begin
        k:=i;
        for j:= (i+1) to dimL do
            if (vector_numeros[j]<vector_numeros[k]) then
                k:=j;
        item:=vector_numeros[k];
        vector_numeros[k]:=vector_numeros[i];
        vector_numeros[i]:=item;
    end;
end;
function buscar_vector_numeros(vector_numeros: t_vector_numeros; num, pri, ult: int8): int8;
var
    medio: int8;
begin
    if (pri<=ult) then

```

```
begin
  medio:=(pri+ult) div 2;
  if (num=vector_numeros[medio]) then
    buscar_vector_numeros:=medio
  else if (num<vector_numeros[medio]) then
    buscar_vector_numeros:=buscar_vector_numeros(vector_numeros,num,pri,medio-1)
  else
    buscar_vector_numeros:=buscar_vector_numeros(vector_numeros,num,medio+1,ult)
  end
else
  buscar_vector_numeros:=-1;
end;
var
  vector_numeros: t_vector_numeros;
  dimL, num, pri, ult, pos: int8;
begin
  randomize;
  dimL:=0;
  cargar_vector_numeros(vector_numeros,dimL);
  if (dimL>0) then
    begin
      imprimir_vector_numeros(vector_numeros,dimL);
      ordenar_vector_numeros(vector_numeros,dimL);
      imprimir_vector_numeros(vector_numeros,dimL);
      num:=1+random(high(int8));
      pri:=1; ult:=dimL;
      pos:=buscar_vector_numeros(vector_numeros,num,pri,ult);
      if (pos<>-1) then
        begin
          textcolor(green); write('El número '); textcolor(yellow); write(num); textcolor(green);
          write(' se encontró en el vector, en la posición '); textcolor(red); write(pos);
        end
      else
        begin
          textcolor(green); write('El número '); textcolor(yellow); write(num); textcolor(green);
          write(' no se encontró en el vector');
        end
      end;
    end;
  end.
end.
```


Ejercicio 6.

Realizar un programa que lea números y que utilice un módulo recursivo que escriba el equivalente en binario de un número decimal. El programa termina cuando el usuario ingresa el número 0 (cero). Ayuda: Analizando las posibilidades, se encuentra que Binario (N) es N si el valor es menor a 2. ¿Cómo se obtienen los dígitos que componen al número? ¿Cómo se achica el número para la próxima llamada recursiva? Ejemplo: si se ingresa 23, el programa debe mostrar 10111.

```
program TP2_E6;
{$codepage UTF8}
uses crt;
const
    num_salida=0;
procedure leer_numero(var num: int8);
begin
    num:=num_salida+random(high(int8));
end;
procedure convertir_binario(num: int16);
var
    digito: int16;
begin
    if (num<>num_salida) then
    begin
        digito:=num mod 2;
        convertir_binario(num div 2);
        write(digito);
    end;
end;
var
    num: int8;
    i: int16;
begin
    randomize;
    i:=0;
    leer_numero(num);
    while (num<>num_salida) do
    begin
        i:=i+1;
        textcolor(green); write(i, '. Número en decimal: '); textcolor(red); writeln(num);
        textcolor(green); write(i, '. Número en binario: '); textcolor(red);
        convertir_binario(num);
        leer_numero(num);
        writeln();
    end;
end.
```

Trabajo Práctico N° 3: **Módulo Imperativo (Árboles 1).**

Ejercicio 1.

Escribir un programa que:

(a) *Implemente un módulo que lea información de socios de un club y las almacene en un árbol binario de búsqueda. De cada socio, se lee número de socio, nombre y edad. La lectura finaliza con el número de socio 0 y el árbol debe quedar ordenado por número de socio.*

(b) *Una vez generado el árbol, realice módulos independientes que reciban el árbol como parámetro y que:*

(i) *Informe el número de socio más grande. Debe invocar a un módulo recursivo que retorne dicho valor.*

(ii) *Informe los datos del socio con el número de socio más chico. Debe invocar a un módulo recursivo que retorne dicho socio.*

(iii) *Informe el número de socio con mayor edad. Debe invocar a un módulo recursivo que retorne dicho valor.*

(iv) *Aumente en 1 la edad de todos los socios.*

(v) *Lea un valor entero e informe si existe o no existe un socio con ese valor. Debe invocar a un módulo recursivo que reciba el valor leído y retorne verdadero o falso.*

(vi) *Lea un nombre e informe si existe o no existe un socio con ese nombre. Debe invocar a un módulo recursivo que reciba el nombre leído y retorne verdadero o falso.*

(vii) *Informe la cantidad de socios. Debe invocar a un módulo recursivo que retorne dicha cantidad.*

(viii) *Informe el promedio de edad de los socios. Debe invocar al módulo recursivo del inciso (vii) e invocar a un módulo recursivo que retorne la suma de las edades de los socios.*

(ix) *Informe, a partir de dos valores que se leen, la cantidad de socios en el árbol cuyo número de socio se encuentra entre los dos valores ingresados. Debe invocar a un módulo recursivo que reciba los dos valores leídos y retorne dicha cantidad.*

(x) *Informe los números de socio en orden creciente.*

(xi) *Informe los números de socio pares en orden decreciente.*

```
program TP3_E1;
{$codepage UTF8}
uses crt;
const
    numero_salida=0;
type
    t_registro_socio=record
        numero: int16;
        nombre: string;
        edad: int8;
    end;
    t_abb_socios=^t_nodo_abb_socios;
    t_nodo_abb_socios=record
```

```
    ele: t_registro_socio;
    hi: t_abb_socios;
    hd: t_abb_socios;
end;
function random_string(length: int8): string;
var
    i: int8;
    string_aux: string;
begin
    string_aux:='';
    for i:= 1 to length do
        string_aux:=string_aux+chr(ord('A')+random(26));
        random_string:=string_aux;
    end;
end;
procedure leer_socio(var registro_socio: t_registro_socio);
var
    i: int8;
begin
    i:=random(100);
    if (i=0) then
        registro_socio.numero:=numero_salida
    else
        registro_socio.numero:=1+random(high(int16));
        if (registro_socio.numero<>numero_salida) then
            begin
                registro_socio.nombre:=random_string(5+random(6));
                registro_socio.edad:=1+random(high(int8)-1);
            end;
        end;
end;
procedure agregar_abb_socios(var abb_socios: t_abb_socios; registro_socio: t_registro_socio);
begin
    if (abb_socios=nil) then
        begin
            new(abb_socios);
            abb_socios^.ele:=registro_socio;
            abb_socios^.hi:=nil;
            abb_socios^.hd:=nil;
        end
    else
        if (registro_socio.numero<=abb_socios^.ele.numero) then
            agregar_abb_socios(abb_socios^.hi,registro_socio)
        else
            agregar_abb_socios(abb_socios^.hd,registro_socio);
        end;
end;
procedure cargar_abb_socios(var abb_socios: t_abb_socios);
var
    registro_socio: t_registro_socio;
begin
    leer_socio(registro_socio);
    while (registro_socio.numero<>numero_salida) do
        begin
            agregar_abb_socios(abb_socios,registro_socio);
            leer_socio(registro_socio);
        end;
    end;
end;
procedure imprimir_registro_socio(registro_socio: t_registro_socio);
begin
    textcolor(green); write('El número de socio del socio es '); textcolor(red);
    writeln(registro_socio.numero);
    textcolor(green); write('El nombre del socio es '); textcolor(red);
    writeln(registro_socio.nombre);
    textcolor(green); write('La edad del socio es '); textcolor(red);
    writeln(registro_socio.edad);
    writeln();
end;
procedure imprimir1_abb_socios(abb_socios: t_abb_socios);
```

```
begin
  if (abb_socios<>nil) then
    begin
      imprimir1_abb_socios(abb_socios^.hi);
      imprimir_registro_socio(abb_socios^.ele);
      imprimir1_abb_socios(abb_socios^.hd);
    end;
  end;
end;
function buscar_mayor_numero(abb_socios: t_abb_socios): int16;
begin
  if (abb_socios^.hd=nil) then
    buscar_mayor_numero:=abb_socios^.ele.numero
  else
    buscar_mayor_numero:=buscar_mayor_numero(abb_socios^.hd);
  end;
end;
function buscar_menor_numero(abb_socios: t_abb_socios): int16;
begin
  if (abb_socios^.hi=nil) then
    buscar_menor_numero:=abb_socios^.ele.numero
  else
    buscar_menor_numero:=buscar_menor_numero(abb_socios^.hi);
  end;
end;
procedure buscar_numero_mayor_edad(abb_socios: t_abb_socios; var edad_max: int8; var
numero_max: int16);
begin
  if (abb_socios<>nil) then
    begin
      buscar_numero_mayor_edad(abb_socios^.hi,edad_max,numero_max);
      if (abb_socios^.ele.edad>edad_max) then
        begin
          edad_max:=abb_socios^.ele.edad;
          numero_max:=abb_socios^.ele.numero;
        end;
      buscar_numero_mayor_edad(abb_socios^.hd,edad_max,numero_max);
    end;
  end;
end;
procedure aumentar_edad(var abb_socios: t_abb_socios);
begin
  if (abb_socios<>nil) then
    begin
      aumentar_edad(abb_socios^.hi);
      abb_socios^.ele.edad:=abb_socios^.ele.edad+1;
      aumentar_edad(abb_socios^.hd);
    end;
  end;
end;
function buscar_numero(abb_socios: t_abb_socios; numero: int16): boolean;
begin
  if (abb_socios=nil) then
    buscar_numero:=false
  else
    if (numero=abb_socios^.ele.numero) then
      buscar_numero:=true
    else if (numero<abb_socios^.ele.numero) then
      buscar_numero:=buscar_numero(abb_socios^.hi,numero)
    else
      buscar_numero:=buscar_numero(abb_socios^.hd,numero);
  end;
end;
function buscar_nombre(abb_socios: t_abb_socios; nombre: string): boolean;
begin
  if (abb_socios=nil) then
    buscar_nombre:=false
  else
    if (nombre=abb_socios^.ele.nombre) then
      buscar_nombre:=true
    else
```

```

        buscar_nombre:=buscar_nombre(abb_socios^.hi,nombre) or
buscar_nombre(abb_socios^.hd,nombre);
end;
function contar_socios1(abb_socios: t_abb_socios): int16;
begin
    if (abb_socios=nil) then
        contar_socios1:=0
    else
        contar_socios1:=contar_socios1(abb_socios^.hi)+contar_socios1(abb_socios^.hd)+1;
    end;
end;
function contar_edades(abb_socios: t_abb_socios): int16;
begin
    if (abb_socios=nil) then
        contar_edades:=0
    else
        contar_edades:=contar_edades(abb_socios^.hi)+contar_edades(abb_socios^.hd)+abb_socios^.ele
.edad;
    end;
end;
function calcular_edad_promedio(abb_socios: t_abb_socios): real;
begin
    calcular_edad_promedio:=contar_edades(abb_socios)/contar_socios1(abb_socios);
end;
procedure verificar_numeros(var numero1, numero2: int16);
var
    aux: int8;
begin
    if (numero1>numero2) then
        begin
            aux:=numero1;
            numero1:=numero2;
            numero2:=aux;
        end;
    end;
end;
function contar_socios2(abb_socios: t_abb_socios; numero1, numero2: int16): int16;
begin
    if (abb_socios=nil) then
        contar_socios2:=0
    else
        if (numero1>=abb_socios^.ele.numero) then
            contar_socios2:=contar_socios2(abb_socios^.hd,numero1,numero2)
        else if (numero2<=abb_socios^.ele.numero) then
            contar_socios2:=contar_socios2(abb_socios^.hi,numero1,numero2)
        else
            contar_socios2:=contar_socios2(abb_socios^.hi,numero1,numero2)+contar_socios2(abb_socios
^.hd,numero1,numero2)+1;
        end;
    end;
end;
procedure imprimir2_abb_socios(abb_socios: t_abb_socios);
begin
    if (abb_socios<>nil) then
        begin
            imprimir2_abb_socios(abb_socios^.hi);
            textcolor(green); write('Número de socio: '); textcolor(red);
writeln(abb_socios^.ele.numero);
            imprimir2_abb_socios(abb_socios^.hd);
        end;
    end;
end;
procedure imprimir3_abb_socios(abb_socios: t_abb_socios);
begin
    if (abb_socios<>nil) then
        begin
            imprimir3_abb_socios(abb_socios^.hd);
            if (abb_socios^.ele.numero mod 2=0) then
                begin
                    textcolor(green); write('Número de socio: '); textcolor(red);
writeln(abb_socios^.ele.numero);
                end;
            end;
        end;
    end;
end;

```

```

    imprimir3_abb_socios(abb_socios^.hi);
end;
end;
var
    abb_socios: t_abb_socios;
    edad_max: int8;
    numero_max, numero, numero1, numero2: int16;
    nombre: string;
begin
    randomize;
    abb_socios:=nil;
    edad_max:=low(int8); numero_max:=numero_salida;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_abb_socios(abb_socios);
    if (abb_socios<>nil) then
    begin
        imprimir1_abb_socios(abb_socios);
        writeln(); textcolor(red); writeln('INCISO (b.i):'); writeln();
        textcolor(green); write('El número de socio más grande es '); textcolor(red);
        writeln(buscar_mayor_numero(abb_socios));
        writeln(); textcolor(red); writeln('INCISO (b.ii):'); writeln();
        textcolor(green); write('El número de socio más chico es '); textcolor(red);
        writeln(buscar_menor_numero(abb_socios));
        writeln(); textcolor(red); writeln('INCISO (b.iii):'); writeln();
        buscar_numero_mayor_edad(abb_socios,edad_max,numero_max);
        textcolor(green); write('El número de socio con mayor edad es '); textcolor(red);
        writeln(numero_max);
        writeln(); textcolor(red); writeln('INCISO (b.iv):'); writeln();
        aumentar_edad(abb_socios);
        imprimir1_abb_socios(abb_socios);
        writeln(); textcolor(red); writeln('INCISO (b.v):'); writeln();
        numero:=1+random(high(int16));
        textcolor(green); write('¿El número de socio '); textcolor(yellow); write(numero);
        textcolor(green); write(' se encuentra en el abb?: '); textcolor(red);
        writeln(buscar_numero(abb_socios,numero));
        writeln(); textcolor(red); writeln('INCISO (b.vi):'); writeln();
        nombre:=random_string(5+random(6));
        textcolor(green); write('¿El nombre de socio '); textcolor(yellow); write(nombre);
        textcolor(green); write(' se encuentra en el abb?: '); textcolor(red);
        writeln(buscar_nombre(abb_socios,nombre));
        writeln(); textcolor(red); writeln('INCISO (b.vii):'); writeln();
        textcolor(green); write('La cantidad de socios es '); textcolor(red);
        writeln(contar_socios1(abb_socios));
        writeln(); textcolor(red); writeln('INCISO (b.viii):'); writeln();
        textcolor(green); write('El promedio de edad de los socios es '); textcolor(red);
        writeln(calcular_edad_promedio(abb_socios):0:2);
        writeln(); textcolor(red); writeln('INCISO (b.ix):'); writeln();
        numero1:=1+random(high(int16)); numero2:=1+random(high(int16));
        verificar_numeros(numero1,numero2);
        textcolor(green); write('La cantidad de socios en el abb cuyo número de socio se encuentra
entre '); textcolor(yellow); write(numero1); textcolor(green); write(' y ');
        textcolor(yellow); write(numero2); textcolor(green); write(' es '); textcolor(red);
        writeln(contar_socios2(abb_socios,numero1,numero2));
        writeln(); textcolor(red); writeln('INCISO (b.x):'); writeln();
        imprimir2_abb_socios(abb_socios);
        writeln(); textcolor(red); writeln('INCISO (b.xi):'); writeln();
        imprimir3_abb_socios(abb_socios);
    end;
end.

```

Ejercicio 2.

Escribir un programa que:

(a) *Implemente un módulo que lea información de ventas de un comercio. De cada venta, se lee código de producto, fecha y cantidad de unidades vendidas. La lectura finaliza con el código de producto 0. Un producto puede estar en más de una venta. Se pide:*

(i) *Generar y retornar un árbol binario de búsqueda de ventas ordenado por código de producto.*

(ii) *Generar y retornar otro árbol binario de búsqueda de productos vendidos ordenado por código de producto. Cada nodo del árbol debe contener el código de producto y la cantidad total de unidades vendidas.*

Nota: El módulo debe retornar los dos árboles.

(b) *Implemente un módulo que reciba el árbol generado en (i) y un código de producto y retorne la cantidad total de unidades vendidas de ese producto.*

(c) *Implemente un módulo que reciba el árbol generado en (ii) y un código de producto y retorne la cantidad total de unidades vendidas de ese producto.*

```
program TP3_E2;
{$codepage UTF8}
uses crt;
const
  codigo_salida=0;
type
  t_registro_venta=record
    codigo: int8;
    fecha: int8;
    cantidad: int8;
  end;
  t_registro_producto=record
    codigo: int8;
    cantidad: int16;
  end;
  t_abb_ventas=^t_nodo_abb_ventas;
  t_nodo_abb_ventas=record
    ele: t_registro_venta;
    hi: t_abb_ventas;
    hd: t_abb_ventas;
  end;
  t_abb_productos=^t_nodo_abb_productos;
  t_nodo_abb_productos=record
    ele: t_registro_producto;
    hi: t_abb_productos;
    hd: t_abb_productos;
  end;
procedure leer_venta(var registro_venta: t_registro_venta);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_venta.codigo:=codigo_salida
  else
    registro_venta.codigo:=1+random(high(int8));
```

```
if (registro_venta.codigo<>codigo_salida) then
begin
    registro_venta.fecha:=1+random(high(int8));
    registro_venta.cantidad:=1+random(high(int8));
end;
end;
procedure agregar_abb_ventas(var abb_ventas: t_abb_ventas; registro_venta: t_registro_venta);
begin
    if (abb_ventas=nil) then
    begin
        new(abb_ventas);
        abb_ventas^.ele:=registro_venta;
        abb_ventas^.hi:=nil;
        abb_ventas^.hd:=nil;
    end
    else
        if (registro_venta.codigo<=abb_ventas^.ele.codigo) then
            agregar_abb_ventas(abb_ventas^.hi,registro_venta)
        else
            agregar_abb_ventas(abb_ventas^.hd,registro_venta);
    end;
end;
procedure cargar_registro_producto(var registro_producto: t_registro_producto; registro_venta:
t_registro_venta);
begin
    registro_producto.codigo:=registro_venta.codigo;
    registro_producto.cantidad:=registro_venta.cantidad;
end;
procedure agregar_abb_productos(var abb_productos: t_abb_productos; registro_venta:
t_registro_venta);
begin
    if (abb_productos=nil) then
    begin
        new(abb_productos);
        cargar_registro_producto(abb_productos^.ele,registro_venta);
        abb_productos^.hi:=nil;
        abb_productos^.hd:=nil;
    end
    else
        if (registro_venta.codigo=abb_productos^.ele.codigo) then
            abb_productos^.ele.cantidad:=abb_productos^.ele.cantidad+registro_venta.cantidad
        else if (registro_venta.codigo<abb_productos^.ele.codigo) then
            agregar_abb_productos(abb_productos^.hi,registro_venta)
        else
            agregar_abb_productos(abb_productos^.hd,registro_venta);
    end;
end;
procedure cargar_abbs(var abb_ventas: t_abb_ventas; var abb_productos: t_abb_productos);
var
    registro_venta: t_registro_venta;
begin
    leer_venta(registro_venta);
    while (registro_venta.codigo<>codigo_salida) do
    begin
        agregar_abb_ventas(abb_ventas,registro_venta);
        agregar_abb_productos(abb_productos,registro_venta);
        leer_venta(registro_venta);
    end;
end;
procedure imprimir_registro_venta(registro_venta: t_registro_venta);
begin
    textcolor(green); write('El código de producto de la venta es '); textcolor(red);
writeln(registro_venta.codigo);
    textcolor(green); write('La fecha de la venta es '); textcolor(red);
writeln(registro_venta.fecha);
    textcolor(green); write('La cantidad de unidades vendidas de la venta es '); textcolor(red);
writeln(registro_venta.cantidad);
writeln();
```



```
end;
procedure imprimir_abb_ventas(abb_ventas: t_abb_ventas);
begin
    if (abb_ventas<>nil) then
        begin
            imprimir_abb_ventas(abb_ventas^.hi);
            imprimir_registro_venta(abb_ventas^.ele);
            imprimir_abb_ventas(abb_ventas^.hd);
        end;
    end;
end;
procedure imprimir_registro_producto(registro_producto: t_registro_producto);
begin
    textcolor(green); write('El código de producto del producto es '); textcolor(red);
    writeln(registro_producto.codigo);
    textcolor(green); write('La cantidad de unidades vendidas del producto es ');
    textcolor(red); writeln(registro_producto.cantidad);
    writeln();
end;
procedure imprimir_abb_productos(abb_productos: t_abb_productos);
begin
    if (abb_productos<>nil) then
        begin
            imprimir_abb_productos(abb_productos^.hi);
            imprimir_registro_producto(abb_productos^.ele);
            imprimir_abb_productos(abb_productos^.hd);
        end;
    end;
end;
function contar_abb_ventas(abb_ventas: t_abb_ventas; codigo: int8): int16;
begin
    if (abb_ventas=nil) then
        contar_abb_ventas:=0
    else
        if (codigo=abb_ventas^.ele.codigo) then
            contar_abb_ventas:=contar_abb_ventas(abb_ventas^.hi,codigo)+abb_ventas^.ele.cantidad
        else if (codigo<abb_ventas^.ele.codigo) then
            contar_abb_ventas:=contar_abb_ventas(abb_ventas^.hi,codigo)
        else
            contar_abb_ventas:=contar_abb_ventas(abb_ventas^.hd,codigo);
    end;
end;
function contar_abb_productos(abb_productos: t_abb_productos; codigo: int8): int16;
begin
    if (abb_productos=nil) then
        contar_abb_productos:=0
    else
        if (codigo=abb_productos^.ele.codigo) then
            contar_abb_productos:=abb_productos^.ele.cantidad
        else if (codigo<abb_productos^.ele.codigo) then
            contar_abb_productos:=contar_abb_productos(abb_productos^.hi,codigo)
        else
            contar_abb_productos:=contar_abb_productos(abb_productos^.hd,codigo);
    end;
end;
var
    abb_ventas: t_abb_ventas;
    abb_productos: t_abb_productos;
    codigo: int8;
begin
    randomize;
    abb_ventas:=nil; abb_productos:=nil;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_abbs(abb_ventas,abb_productos);
    if ((abb_ventas<>nil) and (abb_productos<>nil)) then
        begin
            writeln(); textcolor(red); writeln('ABB_VENTAS:'); writeln();
            imprimir_abb_ventas(abb_ventas);
            writeln(); textcolor(red); writeln('ABB_PRODUCTOS:'); writeln();
            imprimir_abb_productos(abb_productos);
```

```
writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
codigo:=1+random(high(int8));
textcolor(green); write('La cantidad total de unidades vendidas en el abb_ventas del
código de producto '); textcolor(yellow); write(codigo); textcolor(green); write(' es ');
textcolor(red); writeln(contar_abb_ventas(abb_ventas,codigo));
writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
textcolor(green); write('La cantidad total de unidades vendidas en el abb_productos del
código de producto '); textcolor(yellow); write(codigo); textcolor(green); write(' es ');
textcolor(red); write(contar_abb_productos(abb_productos,codigo));
end;
end.
```

Ejercicio 3.

Implementar un programa que contenga:

- (a) Un módulo que lea información de alumnos de Taller de Programación y los almacene en una estructura de datos. De cada alumno, se lee legajo, DNI, año de ingreso y los códigos y notas de los finales rendidos. La estructura generada debe ser eficiente para la búsqueda por número de legajo. La lectura de los alumnos finaliza con legajo 0 y, para cada alumno, el ingreso de las materias finaliza con el código de materia -1.
- (b) Un módulo que reciba la estructura generada en (a) y retorne los DNI y año de ingreso de aquellos alumnos cuyo legajo sea inferior a un valor ingresado como parámetro.
- (c) Un módulo que reciba la estructura generada en (a) y retorne el legajo más grande.
- (d) Un módulo que reciba la estructura generada en (a) y retorne el DNI más grande.
- (e) Un módulo que reciba la estructura generada en (a) y retorne la cantidad de alumnos con legajo impar.
- (f) Un módulo que reciba la estructura generada en (a) y retorne el legajo y el promedio del alumno con mayor promedio.
- (g) Un módulo que reciba la estructura generada en (a) y un valor entero. Este módulo debe retornar los legajos y promedios de los alumnos cuyo promedio supera el valor ingresado.

```
program TP3_E3;
{$codepage UTF8}
uses crt;
const
  nota_ini=1; nota_fin=10;
  legajo_salida=0; codigo_salida=-1;
type
  t_nota=nota_ini..nota_fin;
  t_registro_final=record
    codigo: int8;
    nota: t_nota;
  end;
  t_lista_finales=^t_nodo_finales;
  t_nodo_finales=record
    ele: t_registro_final;
    sig: t_lista_finales;
  end;
  t_registro_alumno1=record
    legajo: int16;
    dni: int32;
    anio_ingreso: int16;
    finales: t_lista_finales;
  end;
  t_abb_alumnos1=^t_nodo_abb_alumnos1;
  t_nodo_abb_alumnos1=record
    ele: t_registro_alumno1;
    hi: t_abb_alumnos1;
    hd: t_abb_alumnos1;
```

```

end;
t_registro_alumno2=record
  dni: int32;
  anio_ingreso: int16;
end;
t_abb_alumnos2:=^t_nodo_abb_alumnos2;
t_nodo_abb_alumnos2=record
  ele: t_registro_alumno2;
  hi: t_abb_alumnos2;
  hd: t_abb_alumnos2;
end;
t_registro_alumno3=record
  legajo: int16;
  promedio: real;
end;
t_abb_alumnos3:=^t_nodo_abb_alumnos3;
t_nodo_abb_alumnos3=record
  ele: t_registro_alumno3;
  hi: t_abb_alumnos3;
  hd: t_abb_alumnos3;
end;
procedure leer_final(var registro_final: t_registro_final);
var
  i: int8;
begin
  i:=random(10);
  if (i=0) then
    registro_final.codigo:=codigo_salida
  else
    registro_final.codigo:=1+random(high(int8));
    if (registro_final.codigo<>codigo_salida) then
      registro_final.nota:=nota_ini+random(nota_fin);
    end;
end;
procedure agregar_adelante_lista_finales(var lista_finales: t_lista_finales; registro_final:
t_registro_final);
var
  nuevo: t_lista_finales;
begin
  new(nuevo);
  nuevo^.ele:=registro_final;
  nuevo^.sig:=lista_finales;
  lista_finales:=nuevo;
end;
procedure leer_finales(var lista_finales: t_lista_finales);
var
  registro_final: t_registro_final;
begin
  leer_final(registro_final);
  while (registro_final.codigo<>codigo_salida) do
    begin
      agregar_adelante_lista_finales(lista_finales,registro_final);
      leer_final(registro_final);
    end;
end;
procedure leer_alumno(var registro_alumno1: t_registro_alumno1);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_alumno1.legajo:=legajo_salida
  else
    registro_alumno1.legajo:=1+random(high(int16));
    if (registro_alumno1.legajo<>legajo_salida) then
      begin
        registro_alumno1.dni:=10000000+random(40000001);

```

```

    registro_alumno1.anio_ingreso:=2000+random(25);
    registro_alumno1.finales:=nil;
    leer_finales(registro_alumno1.finales);
end;
end;
procedure agregar_abb_alumnos1(var abb_alumnos1: t_abb_alumnos1; registro_alumno1:
t_registro_alumno1);
begin
    if (abb_alumnos1=nil) then
    begin
        new(abb_alumnos1);
        abb_alumnos1^.ele:=registro_alumno1;
        abb_alumnos1^.hi:=nil;
        abb_alumnos1^.hd:=nil;
    end
    else
        if (registro_alumno1.legajo<=abb_alumnos1^.ele.legajo) then
            agregar_abb_alumnos1(abb_alumnos1^.hi,registro_alumno1)
        else
            agregar_abb_alumnos1(abb_alumnos1^.hd,registro_alumno1);
        end;
    end;
end;
procedure cargar_abb_alumnos1(var abb_alumnos1: t_abb_alumnos1);
var
    registro_alumno1: t_registro_alumno1;
begin
    leer_alumno(registro_alumno1);
    while (registro_alumno1.legajo<>legajo_salida) do
    begin
        agregar_abb_alumnos1(abb_alumnos1,registro_alumno1);
        leer_alumno(registro_alumno1);
    end;
end;
procedure imprimir_registro_final(registro_final: t_registro_final; legajo, final: int16);
begin
    textcolor(green); write('El código del final '); textcolor(yellow); write(final);
    textcolor(green); write(' del legajo '); textcolor(yellow); write(legajo); textcolor(green);
    write(' es '); textcolor(red); writeln(registro_final.codigo);
    textcolor(green); write('La nota del final '); textcolor(yellow); write(final);
    textcolor(green); write(' del legajo '); textcolor(yellow); write(legajo); textcolor(green);
    write(' es '); textcolor(red); writeln(registro_final.nota);
end;
procedure imprimir_lista_finales(lista_finales: t_lista_finales; legajo: int16);
var
    i: int16;
begin
    i:=0;
    while (lista_finales<>nil) do
    begin
        i:=i+1;
        imprimir_registro_final(lista_finales^.ele,legajo,i);
        lista_finales:=lista_finales^.sig;
    end;
end;
procedure imprimir_registro_alumno1(registro_alumno1: t_registro_alumno1);
begin
    textcolor(green); write('El legajo del alumno es '); textcolor(red);
    writeln(registro_alumno1.legajo);
    textcolor(green); write('El DNI del alumno es '); textcolor(red);
    writeln(registro_alumno1.dni);
    textcolor(green); write('El año de ingreso del alumno es '); textcolor(red);
    writeln(registro_alumno1.anio_ingreso);
    imprimir_lista_finales(registro_alumno1.finales,registro_alumno1.legajo);
    writeln();
end;
procedure imprimir_abb_alumnos1(abb_alumnos1: t_abb_alumnos1);
begin

```

```

    if (abb_alumnos1<>nil) then
    begin
        imprimir_abb_alumnos1(abb_alumnos1^.hi);
        imprimir_registro_alumno1(abb_alumnos1^.ele);
        imprimir_abb_alumnos1(abb_alumnos1^.hd);
    end;
end;
procedure cargar_registro_alumno2(var registro_alumno2: t_registro_alumno2; registro_alumno1:
t_registro_alumno1);
begin
    registro_alumno2.dni:=registro_alumno1.dni;
    registro_alumno2.anio_ingreso:=registro_alumno1.anio_ingreso;
end;
procedure agregar_abb_alumnos2(var abb_alumnos2: t_abb_alumnos2; registro_alumno1:
t_registro_alumno1);
begin
    if (abb_alumnos2=nil) then
    begin
        new(abb_alumnos2);
        cargar_registro_alumno2(abb_alumnos2^.ele,registro_alumno1);
        abb_alumnos2^.hi:=nil;
        abb_alumnos2^.hd:=nil;
    end
    else
        if (registro_alumno1.dni<=abb_alumnos2^.ele.dni) then
            agregar_abb_alumnos2(abb_alumnos2^.hi,registro_alumno1)
        else
            agregar_abb_alumnos2(abb_alumnos2^.hd,registro_alumno1);
    end;
end;
procedure cargar_abb_alumnos2(var abb_alumnos2: t_abb_alumnos2; abb_alumnos1: t_abb_alumnos1;
legajo: int16);
begin
    if (abb_alumnos1<>nil) then
    begin
        if (abb_alumnos1^.ele.legajo<legajo) then
        begin
            cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.hi,legajo);
            agregar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.ele);
            cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.hd,legajo);
        end
        else
            cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.hi,legajo);
        end;
    end;
end;
procedure imprimir_registro_alumno2(registro_alumno2: t_registro_alumno2);
begin
    textcolor(green); write('El DNI del alumno es '); textcolor(red);
writeln(registro_alumno2.dni);
    textcolor(green); write('El año de ingreso del alumno es '); textcolor(red);
writeln(registro_alumno2.anio_ingreso);
    writeln();
end;
procedure imprimir_abb_alumnos2(abb_alumnos2: t_abb_alumnos2);
begin
    if (abb_alumnos2<>nil) then
    begin
        imprimir_abb_alumnos2(abb_alumnos2^.hi);
        imprimir_registro_alumno2(abb_alumnos2^.ele);
        imprimir_abb_alumnos2(abb_alumnos2^.hd);
    end;
end;
function buscar_mayor_legajo(abb_alumnos1: t_abb_alumnos1): int16;
begin
    if (abb_alumnos1^.hd=nil) then
        buscar_mayor_legajo:=abb_alumnos1^.ele.legajo
    else

```

```
        buscar_mayor_legajo:=buscar_mayor_legajo(abb_alumnos1^.hd);
    end;
procedure buscar_mayor_dni(abb_alumnos1: t_abb_alumnos1; var dni_max: int32);
begin
    if (abb_alumnos1<>nil) then
        begin
            buscar_mayor_dni(abb_alumnos1^.hi,dni_max);
            if (abb_alumnos1^.ele.dni>dni_max) then
                dni_max:=abb_alumnos1^.ele.dni;
            buscar_mayor_dni(abb_alumnos1^.hd,dni_max);
        end;
    end;
end;
procedure contar_legajos_impar(abb_alumnos1: t_abb_alumnos1; var legajos_impar: int16);
begin
    if (abb_alumnos1<>nil) then
        begin
            contar_legajos_impar(abb_alumnos1^.hi,legajos_impar);
            if (abb_alumnos1^.ele.legajo mod 2<>0) then
                legajos_impar:=legajos_impar+1;
            contar_legajos_impar(abb_alumnos1^.hd,legajos_impar);
        end;
    end;
end;
function calcular_promedio(lista_finales: t_lista_finales): real;
var
    notas_total, notas: int16;
begin
    notas_total:=0; notas:=0;
    while (lista_finales<>nil) do
        begin
            notas_total:=notas_total+lista_finales^.ele.nota;
            notas:=notas+1;
            lista_finales:=lista_finales^.sig;
        end;
    if (notas>0) then
        calcular_promedio:=notas_total/notas
    else
        calcular_promedio:=notas_total;
    end;
end;
procedure buscar_legajo_mayor_promedio(abb_alumnos1: t_abb_alumnos1; var promedio_max: real;
var legajo_max: int16);
var
    promedio: real;
begin
    if (abb_alumnos1<>nil) then
        begin
            buscar_legajo_mayor_promedio(abb_alumnos1^.hi,promedio_max,legajo_max);
            promedio:=calcular_promedio(abb_alumnos1^.ele.finales);
            if (promedio>promedio_max) then
                begin
                    promedio_max:=promedio;
                    legajo_max:=abb_alumnos1^.ele.legajo;
                end;
            buscar_legajo_mayor_promedio(abb_alumnos1^.hd,promedio_max,legajo_max);
        end;
    end;
end;
procedure cargar_registro_alumno3(var registro_alumno3: t_registro_alumno3; legajo: int16;
promedio_alumno: real);
begin
    registro_alumno3.legajo:=legajo;
    registro_alumno3.promedio:=promedio_alumno;
end;
end;
procedure agregar_abb_alumnos3(var abb_alumnos3: t_abb_alumnos3; legajo: int16;
promedio_alumno: real);
begin
    if (abb_alumnos3=nil) then
        begin
```

```

    new(abb_alumnos3);
    cargar_registro_alumno3(abb_alumnos3^.ele,legajo,promedio_alumno);
    abb_alumnos3^.hi:=nil;
    abb_alumnos3^.hd:=nil;
end
else
    if (legajo<=abb_alumnos3^.ele.legajo) then
        agregar_abb_alumnos3(abb_alumnos3^.hi,legajo,promedio_alumno)
    else
        agregar_abb_alumnos3(abb_alumnos3^.hd,legajo,promedio_alumno);
end;
procedure cargar_abb_alumnos3(var abb_alumnos3: t_abb_alumnos3; abb_alumnos1: t_abb_alumnos1;
promedio: real);
var
    promedio_alumno: real;
begin
    if (abb_alumnos1<>nil) then
        begin
            cargar_abb_alumnos3(abb_alumnos3,abb_alumnos1^.hi,promedio);
            promedio_alumno:=calcular_promedio(abb_alumnos1^.ele.finales);
            if (promedio_alumno>promedio) then
                agregar_abb_alumnos3(abb_alumnos3,abb_alumnos1^.ele.legajo,promedio_alumno);
            cargar_abb_alumnos3(abb_alumnos3,abb_alumnos1^.hd,promedio);
        end;
    end;
procedure imprimir_registro_alumno3(registro_alumno3: t_registro_alumno3);
begin
    textcolor(green); write('El legajo del alumno es '); textcolor(red);
    writeln(registro_alumno3.legajo);
    textcolor(green); write('El promedio del alumno es '); textcolor(red);
    writeln(registro_alumno3.promedio:0:2);
    writeln();
end;
procedure imprimir_abb_alumnos3(abb_alumnos3: t_abb_alumnos3);
begin
    if (abb_alumnos3<>nil) then
        begin
            imprimir_abb_alumnos3(abb_alumnos3^.hi);
            imprimir_registro_alumno3(abb_alumnos3^.ele);
            imprimir_abb_alumnos3(abb_alumnos3^.hd);
        end;
    end;
var
    abb_alumnos1: t_abb_alumnos1;
    abb_alumnos2: t_abb_alumnos2;
    abb_alumnos3: t_abb_alumnos3;
    legajo, legajos_impar, legajo_max: int16;
    dni_max: int32;
    promedio_max, promedio: real;
begin
    randomize;
    abb_alumnos1:=nil;
    abb_alumnos2:=nil;
    dni_max:=low(int32);
    legajos_impar:=0;
    promedio_max:=-9999999; legajo_max:=0;
    abb_alumnos3:=nil;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_abb_alumnos1(abb_alumnos1);
    if (abb_alumnos1<>nil) then
        begin
            imprimir_abb_alumnos1(abb_alumnos1);
            writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
            legajo:=1+random(high(int16));
            cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1,legajo);
            if (abb_alumnos2<>nil) then

```



```
    imprimir_abb_alumnos2(abb_alumnos2);
    writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
    textcolor(green); write('El legajo más grande es '); textcolor(red);
writeln(buscar_mayor_legajo(abb_alumnos1));
    writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
    buscar_mayor_dni(abb_alumnos1,dni_max);
    textcolor(green); write('El DNI más grande es '); textcolor(red); writeln(dni_max);
    writeln(); textcolor(red); writeln('INCISO (e):'); writeln();
    contar_legajos_impar(abb_alumnos1,legajos_impar);
    textcolor(green); write('La cantidad de alumnos con legajo impar es '); textcolor(red);
writeln(legajos_impar);
    writeln(); textcolor(red); writeln('INCISO (f):'); writeln();
    buscar_legajo_mayor_promedio(abb_alumnos1,promedio_max,legajo_max);
    textcolor(green); write('El legajo y el promedio del alumno con mayor promedio son ');
textcolor(red); write(legajo_max); textcolor(green); write(' y '); textcolor(red);
write(promedio_max:0:2); textcolor(green); writeln(', respectivamente');
    writeln(); textcolor(red); writeln('INCISO (g):'); writeln();
    promedio:=1+random(91)/10;
    cargar_abb_alumnos3(abb_alumnos3,abb_alumnos1,promedio);
    if (abb_alumnos3<>nil) then
        imprimir_abb_alumnos3(abb_alumnos3);
    end;
end.
```

Trabajo Práctico N° 4: **Módulo Imperativo (Árboles 2).**

Ejercicio 1.

Implementar un programa modularizado para una librería que:

(a) Almacene los productos vendidos en una estructura eficiente para la búsqueda por código de producto. De cada producto, deben quedar almacenados la cantidad total de unidades vendidas y el monto total. De cada venta, se lee código de venta, código del producto vendido, cantidad de unidades vendidas y precio unitario. El ingreso de las ventas finaliza cuando se lee el código de venta -1.

(b) Imprima el contenido del árbol ordenado por código de producto.

(c) Contenga un módulo que reciba la estructura generada en el inciso (a) y retorne el código de producto con mayor cantidad de unidades vendidas.

(d) Contenga un módulo que reciba la estructura generada en el inciso (a) y un código de producto y retorne la cantidad de códigos menores que él que hay en la estructura.

(e) Contenga un módulo que reciba la estructura generada en el inciso (a) y dos códigos de producto y retorne el monto total entre todos los códigos de productos comprendidos entre los dos valores recibidos (sin incluir).

```
program TP4_E1;
{$codepage UTF8}
uses crt;
const
  codigo_venta_salida=-1;
type
  t_registro_venta=record
    codigo_venta: int16;
    codigo_producto: int16;
    cantidad: int8;
    precio: real;
  end;
  t_registro_producto=record
    codigo_producto: int16;
    cantidad_total: int16;
    monto_total: real;
  end;
  t_abb_productos=^t_nodo_abb_productos;
  t_nodo_abb_productos=record
    ele: t_registro_producto;
    hi: t_abb_productos;
    hd: t_abb_productos;
  end;
procedure leer_venta(var registro_venta: t_registro_venta);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_venta.codigo_venta:=codigo_venta_salida
  else
```

```

    registro_venta.codigo_venta:=random(high(int16));
    if (registro_venta.codigo_venta<>codigo_venta_salida) then
    begin
        registro_venta.codigo_producto:=1+random(high(int16));
        registro_venta.cantidad:=1+random(high(int8));
        registro_venta.precio:=1+random(100);
    end;
end;
procedure cargar_registro_producto(var registro_producto: t_registro_producto; registro_venta:
t_registro_venta);
begin
    registro_producto.codigo_producto:=registro_venta.codigo_producto;
    registro_producto.cantidad_total:=registro_venta.cantidad;
    registro_producto.monto_total:=registro_venta.cantidad*registro_venta.precio;
end;
procedure agregar_abb_productos(var abb_productos: t_abb_productos; registro_venta:
t_registro_venta);
begin
    if (abb_productos=nil) then
    begin
        new(abb_productos);
        cargar_registro_producto(abb_productos^.ele,registro_venta);
        abb_productos^.hi:=nil;
        abb_productos^.hd:=nil;
    end
    else
        if (registro_venta.codigo_producto=abb_productos^.ele.codigo_producto) then
        begin
            abb_productos^.ele.cantidad_total:=abb_productos^.ele.cantidad_total+registro_venta.cant
idad;
            abb_productos^.ele.monto_total:=abb_productos^.ele.monto_total+registro_venta.cantidad*r
egistro_venta.precio;
        end
        else
            if (registro_venta.codigo_producto<abb_productos^.ele.codigo_producto) then
                agregar_abb_productos(abb_productos^.hi,registro_venta)
            else
                agregar_abb_productos(abb_productos^.hd,registro_venta);
        end;
end;
procedure cargar_abb_productos(var abb_productos: t_abb_productos);
var
    registro_venta: t_registro_venta;
begin
    leer_venta(registro_venta);
    while (registro_venta.codigo_venta<>codigo_venta_salida) do
    begin
        agregar_abb_productos(abb_productos,registro_venta);
        leer_venta(registro_venta);
    end;
end;
procedure imprimir_registro_producto(registro_producto: t_registro_producto);
begin
    textcolor(green); write('El código de producto del producto es '); textcolor(red);
writeln(registro_producto.codigo_producto);
    textcolor(green); write('La cantidad total de unidades vendidas del producto es ');
textcolor(red); writeln(registro_producto.cantidad_total);
    textcolor(green); write('El monto total del producto es $'); textcolor(red);
writeln(registro_producto.monto_total:0:2);
    writeln();
end;
procedure imprimir_abb_productos(abb_productos: t_abb_productos);
begin
    if (abb_productos<>nil) then
    begin
        imprimir_abb_productos(abb_productos^.hi);
        imprimir_registro_producto(abb_productos^.ele);
    end;
end;

```

```

    imprimir_abb_productos(abb_productos^.hd);
end;
end;
procedure buscar_codigo_mayor_cantidad(abb_productos: t_abb_productos; var cantidad_max,
codigo_max: int16);
begin
    if (abb_productos<>nil) then
    begin
        buscar_codigo_mayor_cantidad(abb_productos^.hi,cantidad_max,codigo_max);
        if (abb_productos^.ele.cantidad_total>cantidad_max) then
        begin
            cantidad_max:=abb_productos^.ele.cantidad_total;
            codigo_max:=abb_productos^.ele.codigo_producto;
        end;
        buscar_codigo_mayor_cantidad(abb_productos^.hd,cantidad_max,codigo_max);
    end;
end;
function contar_codigos(abb_productos: t_abb_productos; codigo: int16): int16;
begin
    if (abb_productos=nil) then
        contar_codigos:=0
    else
        if (abb_productos^.ele.codigo_producto<codigo) then
            contar_codigos:=contar_codigos(abb_productos^.hi,codigo)+contar_codigos(abb_productos^.hd,codigo)+1
        else
            contar_codigos:=contar_codigos(abb_productos^.hi,codigo);
        end;
    end;
end;
procedure verificar_codigos(var codigo1, codigo2: int16);
var
    aux: int16;
begin
    if (codigo1>codigo2) then
    begin
        aux:=codigo1;
        codigo1:=codigo2;
        codigo2:=aux;
    end;
end;
function contar_monto_total(abb_productos: t_abb_productos; codigo1, codigo2: int16): real;
begin
    if (abb_productos=nil) then
        contar_monto_total:=0
    else
        if (codigo1>=abb_productos^.ele.codigo_producto) then
            contar_monto_total:=contar_monto_total(abb_productos^.hd,codigo1,codigo2)
        else if (codigo2<=abb_productos^.ele.codigo_producto) then
            contar_monto_total:=contar_monto_total(abb_productos^.hi,codigo1,codigo2)
        else
            contar_monto_total:=contar_monto_total(abb_productos^.hi,codigo1,codigo2)+contar_monto_total(abb_productos^.hd,codigo1,codigo2)+abb_productos^.ele.monto_total;
        end;
    end;
end;
var
    abb_productos: t_abb_productos;
    cantidad_max, codigo_max, codigo, codigo1, codigo2: int16;
begin
    randomize;
    abb_productos:=nil;
    cantidad_max:=low(int16); codigo_max:=0;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_abb_productos(abb_productos);
    if (abb_productos<>nil) then
    begin
        writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
        imprimir_abb_productos(abb_productos);
        writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
    end;
end;

```

```
    buscar_codigo_mayor_cantidad(abb_productos,cantidad_max,codigo_max);
    textcolor(green); write('El código de producto con mayor cantidad de unidades vendidas es
'); textcolor(red); writeln(codigo_max);
    writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
    codigo:=1+random(high(int16));
    textcolor(green); write('La cantidad de códigos menores que el código de producto ');
textcolor(yellow); write(codigo); textcolor(green); write(' es '); textcolor(red);
writeln(contar_codigos(abb_productos,codigo));
    writeln(); textcolor(red); writeln('INCISO (e):'); writeln();
    codigo1:=1+random(high(int16)); codigo2:=1+random(high(int16));
    verificar_codigos(codigo1,codigo2);
    textcolor(green); write('El monto total en el abb cuyo código de producto se encuentra
entre '); textcolor(yellow); write(codigo1); textcolor(green); write(' y ');
textcolor(yellow); write(codigo2); textcolor(green); write(' es $'); textcolor(red);
write(contar_monto_total(abb_productos,codigo1,codigo2):0:2);
    end;
end.
```

Ejercicio 2.

Una biblioteca nos ha encargado procesar la información de los préstamos realizados durante el año 2021. De cada préstamo, se conoce el ISBN del libro, el número de socio, día y mes del préstamo y cantidad de días prestados. Implementar un programa con:

(a) Un módulo que lea préstamos y retorne 2 estructuras de datos con la información de los préstamos. La lectura de los préstamos finaliza con ISBN -1. Las estructuras deben ser eficientes para buscar por ISBN.

(i) En una estructura, cada préstamo debe estar en un nodo.

(ii) En otra estructura, cada nodo debe contener todos los préstamos realizados al ISBN (prestar atención sobre los datos que se almacenan).

(b) Un módulo recursivo que reciba la estructura generada en (i) y retorne el ISBN más grande.

(c) Un módulo recursivo que reciba la estructura generada en (ii) y retorne el ISBN más pequeño.

(d) Un módulo recursivo que reciba la estructura generada en (i) y un número de socio. El módulo debe retornar la cantidad de préstamos realizados a dicho socio.

(e) Un módulo recursivo que reciba la estructura generada en (ii) y un número de socio. El módulo debe retornar la cantidad de préstamos realizados a dicho socio.

(f) Un módulo que reciba la estructura generada en (i) y retorne una nueva estructura ordenada ISBN, donde cada ISBN aparezca una vez junto a la cantidad total de veces que se prestó.

(g) Un módulo que reciba la estructura generada en (ii) y retorne una nueva estructura ordenada ISBN, donde cada ISBN aparezca una vez junto a la cantidad total de veces que se prestó.

(h) Un módulo recursivo que reciba la estructura generada en (g) y muestre su contenido.

(i) Un módulo recursivo que reciba la estructura generada en (i) y dos valores de ISBN. El módulo debe retornar la cantidad total de préstamos realizados a los ISBN comprendidos entre los dos valores recibidos (incluidos).

(j) Un módulo recursivo que reciba la estructura generada en (ii) y dos valores de ISBN. El módulo debe retornar la cantidad total de préstamos realizados a los ISBN comprendidos entre los dos valores recibidos (incluidos).

```
program TP4_E2;
{$codepage UTF8}
uses crt;
const
  dia_ini=1; dia_fin=31;
  mes_ini=1; mes_fin=12;
  isbn_salida=-1;
```

```

type
  t_dia=dia_ini..dia_fin;
  t_mes=mес_ini..mes_fin;
  t_registro_prestamo1=record
    isbn: int8;
    socio: int8;
    dia: t_dia;
    mes: t_mes;
    dias_prestados: int8;
  end;
  t_abb_prestamos=^t_nodo_abb_prestamos;
  t_nodo_abb_prestamos=record
    ele: t_registro_prestamo1;
    hi: t_abb_prestamos;
    hd: t_abb_prestamos;
  end;
  t_registro_prestamo2=record
    socio: int8;
    dia: t_dia;
    mes: t_mes;
    dias_prestados: int8;
  end;
  t_lista_prestamos=^t_nodo_prestamos;
  t_nodo_prestamos=record
    ele: t_registro_prestamo2;
    sig: t_lista_prestamos;
  end;
  t_registro_isbn1=record
    isbn: int8;
    prestamos: t_lista_prestamos;
  end;
  t_abb_isbns=^t_nodo_abb_isbns;
  t_nodo_abb_isbns=record
    ele: t_registro_isbn1;
    hi: t_abb_isbns;
    hd: t_abb_isbns;
  end;
  t_registro_isbn2=record
    isbn: int8;
    prestamos: int16;
  end;
  t_lista_isbns=^t_nodo_isbns;
  t_nodo_isbns=record
    ele: t_registro_isbn2;
    sig: t_lista_isbns;
  end;
procedure leer_prestamo(var registro_prestamo1: t_registro_prestamo1);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_prestamo1.isbn:=isbn_salida
  else
    registro_prestamo1.isbn:=1+random(high(int8));
    if (registro_prestamo1.isbn<>isbn_salida) then
      begin
        registro_prestamo1.socio:=1+random(high(int8));
        registro_prestamo1.dia:=dia_ini+random(dia_fin);
        registro_prestamo1.mes:=mes_ini+random(mes_fin);
        registro_prestamo1.dias_prestados:=1+random(high(int8));
      end;
  end;
end;
procedure agregar_abb_prestamos(var abb_prestamos: t_abb_prestamos; registro_prestamo1:
t_registro_prestamo1);
begin

```

```
if (abb_prestamos=nil) then
begin
  new(abb_prestamos);
  abb_prestamos^.ele:=registro_prestamo1;
  abb_prestamos^.hi:=nil;
  abb_prestamos^.hd:=nil;
end
else
  if (registro_prestamo1.isbn<=abb_prestamos^.ele.isbn) then
    agregar_abb_prestamos(abb_prestamos^.hi,registro_prestamo1)
  else
    agregar_abb_prestamos(abb_prestamos^.hd,registro_prestamo1);
end;
procedure cargar_registro_prestamo2(var registro_prestamo2: t_registro_prestamo2;
registro_prestamo1: t_registro_prestamo1);
begin
  registro_prestamo2.socio:=registro_prestamo1.socio;
  registro_prestamo2.dia:=registro_prestamo1.dia;
  registro_prestamo2.mes:=registro_prestamo1.mes;
  registro_prestamo2.dias_prestados:=registro_prestamo1.dias_prestados;
end;
procedure agregar_adelante_lista_prestamos(var lista_prestamos: t_lista_prestamos;
registro_prestamo1: t_registro_prestamo1);
var
  nuevo: t_lista_prestamos;
begin
  new(nuevo);
  cargar_registro_prestamo2(nuevo^.ele,registro_prestamo1);
  nuevo^.sig:=lista_prestamos;
  lista_prestamos:=nuevo;
end;
procedure cargar_registro_isbn1(var registro_isbn1: t_registro_isbn1; registro_prestamo1:
t_registro_prestamo1);
begin
  registro_isbn1.isbn:=registro_prestamo1.isbn;
  registro_isbn1.prestamos:=nil;
  agregar_adelante_lista_prestamos(registro_isbn1.prestamos,registro_prestamo1);
end;
procedure agregar_abb_isbns(var abb_isbns: t_abb_isbns; registro_prestamo1:
t_registro_prestamo1);
begin
  if (abb_isbns=nil) then
  begin
    new(abb_isbns);
    cargar_registro_isbn1(abb_isbns^.ele,registro_prestamo1);
    abb_isbns^.hi:=nil;
    abb_isbns^.hd:=nil;
  end
  else
    if (registro_prestamo1.isbn=abb_isbns^.ele.isbn) then
      agregar_adelante_lista_prestamos(abb_isbns^.ele.prestamos,registro_prestamo1)
    else if (registro_prestamo1.isbn<abb_isbns^.ele.isbn) then
      agregar_abb_isbns(abb_isbns^.hi,registro_prestamo1)
    else
      agregar_abb_isbns(abb_isbns^.hd,registro_prestamo1);
  end;
end;
procedure cargar_abbs(var abb_prestamos: t_abb_prestamos; var abb_isbns: t_abb_isbns);
var
  registro_prestamo1: t_registro_prestamo1;
begin
  leer_prestamo(registro_prestamo1);
  while (registro_prestamo1.isbn<>isbn_salida) do
  begin
    agregar_abb_prestamos(abb_prestamos,registro_prestamo1);
    agregar_abb_isbns(abb_isbns,registro_prestamo1);
    leer_prestamo(registro_prestamo1);
```



```

    end;
end;
procedure imprimir_registro_prestamo1(registro_prestamo1: t_registro_prestamo1);
begin
    textcolor(green); write('El ISBN del préstamo es '); textcolor(red);
writeln(registro_prestamo1.isbn);
    textcolor(green); write('El número de socio del préstamo es '); textcolor(red);
writeln(registro_prestamo1.socio);
    textcolor(green); write('El día del préstamo es '); textcolor(red);
writeln(registro_prestamo1.dia);
    textcolor(green); write('El mes del préstamo es '); textcolor(red);
writeln(registro_prestamo1.mes);
    textcolor(green); write('La cantidad de días prestados del préstamo es '); textcolor(red);
writeln(registro_prestamo1.dias_prestados);
    writeln();
end;
procedure imprimir_abb_prestamos(abb_prestamos: t_abb_prestamos);
begin
    if (abb_prestamos<>nil) then
    begin
        imprimir_abb_prestamos(abb_prestamos^.hi);
        imprimir_registro_prestamo1(abb_prestamos^.ele);
        imprimir_abb_prestamos(abb_prestamos^.hd);
    end;
end;
procedure imprimir_registro_prestamo2(registro_prestamo2: t_registro_prestamo2; isbn: int8;
prestamo: int16);
begin
    textcolor(green); write('El número de socio del préstamo '); textcolor(yellow);
write(prestamo); textcolor(green); write(' del ISBN '); textcolor(yellow); write(isbn);
textcolor(green); write(' es '); textcolor(red); writeln(registro_prestamo2.socio);
    textcolor(green); write('El día del préstamo '); textcolor(yellow); write(prestamo);
textcolor(green); write(' del ISBN '); textcolor(yellow); write(isbn); textcolor(green);
write(' es '); textcolor(red); writeln(registro_prestamo2.dia);
    textcolor(green); write('El mes del préstamo '); textcolor(yellow); write(prestamo);
textcolor(green); write(' del ISBN '); textcolor(yellow); write(isbn); textcolor(green);
write(' es '); textcolor(red); writeln(registro_prestamo2.mes);
    textcolor(green); write('La cantidad de días prestados del préstamo '); textcolor(yellow);
write(prestamo); textcolor(green); write(' del ISBN '); textcolor(yellow); write(isbn);
textcolor(green); write(' es '); textcolor(red); writeln(registro_prestamo2.dias_prestados);
end;
procedure imprimir_lista_prestamos(lista_prestamos: t_lista_prestamos; isbn: int8);
var
    i: int16;
begin
    i:=0;
    while (lista_prestamos<>nil) do
    begin
        i:=i+1;
        imprimir_registro_prestamo2(lista_prestamos^.ele,isbn,i);
        lista_prestamos:=lista_prestamos^.sig;
    end;
end;
procedure imprimir_registro_isbn1(registro_isbn1: t_registro_isbn1);
begin
    textcolor(green); write('El ISBN del préstamo es '); textcolor(red);
writeln(registro_isbn1.isbn);
    imprimir_lista_prestamos(registro_isbn1.prestamos,registro_isbn1.isbn);
    writeln();
end;
procedure imprimir_abb_isbns(abb_isbns: t_abb_isbns);
begin
    if (abb_isbns<>nil) then
    begin
        imprimir_abb_isbns(abb_isbns^.hi);
        imprimir_registro_isbn1(abb_isbns^.ele);
    end;
end;

```

```

    imprimir_abb_isbns(abb_isbns^.hd);
end;
end;
function buscar_mayor_isbn(abb_prestamos: t_abb_prestamos): int8;
begin
    if (abb_prestamos^.hd=nil) then
        buscar_mayor_isbn:=abb_prestamos^.ele.isbn
    else
        buscar_mayor_isbn:=buscar_mayor_isbn(abb_prestamos^.hd);
    end;
end;
function buscar_menor_isbn(abb_isbns: t_abb_isbns): int8;
begin
    if (abb_isbns^.hi=nil) then
        buscar_menor_isbn:=abb_isbns^.ele.isbn
    else
        buscar_menor_isbn:=buscar_menor_isbn(abb_isbns^.hi);
    end;
end;
function contar_abb_prestamos(abb_prestamos: t_abb_prestamos; socio: int8): int16;
begin
    if (abb_prestamos=nil) then
        contar_abb_prestamos:=0
    else
        if (socio=abb_prestamos^.ele.socio) then
            contar_abb_prestamos:=contar_abb_prestamos(abb_prestamos^.hi,socio)+contar_abb_prestamos
(abb_prestamos^.hd,socio)+1
        else
            contar_abb_prestamos:=contar_abb_prestamos(abb_prestamos^.hi,socio)+contar_abb_prestamos
(abb_prestamos^.hd,socio);
        end;
    end;
end;
function contar_socios(lista_prestamos: t_lista_prestamos; socio: int8): int16;
var
    socios: int16;
begin
    socios:=0;
    while (lista_prestamos<>nil) do
        begin
            if (socio=lista_prestamos^.ele.socio) then
                socios:=socios+1;
            lista_prestamos:=lista_prestamos^.sig;
            end;
        contar_socios:=socios;
    end;
end;
function contar_abb_isbns(abb_isbns: t_abb_isbns; socio: int8): int16;
begin
    if (abb_isbns=nil) then
        contar_abb_isbns:=0
    else
        contar_abb_isbns:=contar_abb_isbns(abb_isbns^.hi,socio)+contar_abb_isbns(abb_isbns^.hd,socio)+
contar_socios(abb_isbns^.ele.prestamos,socio);
    end;
end;
procedure cargar1_registro_isbn2(var registro_isbn2: t_registro_isbn2; isbn: int8);
begin
    registro_isbn2.isbn:=isbn;
    registro_isbn2.prestamos:=1;
end;
procedure agregar_adelante_lista_isbns1(var lista_isbns1: t_lista_isbns; isbn: int8);
var
    nuevo: t_lista_isbns;
begin
    new(nuevo);
    cargar1_registro_isbn2(nuevo^.ele,isbn);
    nuevo^.sig:=lista_isbns1;
    lista_isbns1:=nuevo;
end;
procedure cargar_lista_isbns1(var lista_isbns1: t_lista_isbns; abb_prestamos:
t_abb_prestamos);

```

```

begin
  if (abb_prestamos<>nil) then
    begin
      cargar_lista_isbns1(lista_isbns1,abb_prestamos^.hd);
      if ((lista_isbns1<>nil) and (lista_isbns1^.ele.isbn=abb_prestamos^.ele.isbn)) then
        lista_isbns1^.ele.prestamos:=lista_isbns1^.ele.prestamos+1
      else
        agregar_adelante_lista_isbns1(lista_isbns1,abb_prestamos^.ele.isbn);
        cargar_lista_isbns1(lista_isbns1,abb_prestamos^.hi);
      end;
    end;
end;
function contar_prestamos(lista_prestamos: t_lista_prestamos): int16;
var
  prestamos: int16;
begin
  prestamos:=0;
  while (lista_prestamos<>nil) do
    begin
      prestamos:=prestamos+1;
      lista_prestamos:=lista_prestamos^.sig;
    end;
  contar_prestamos:=prestamos;
end;
procedure cargar2_registro_isbn2(var registro_isbn2: t_registro_isbn2; registro_isbn1:
t_registro_isbn1);
begin
  registro_isbn2.isbn:=registro_isbn1.isbn;
  registro_isbn2.prestamos:=contar_prestamos(registro_isbn1.prestamos);
end;
procedure agregar_adelante_lista_isbns2(var lista_isbns2: t_lista_isbns; registro_isbn1:
t_registro_isbn1);
var
  nuevo: t_lista_isbns;
begin
  new(nuevo);
  cargar2_registro_isbn2(nuevo^.ele,registro_isbn1);
  nuevo^.sig:=lista_isbns2;
  lista_isbns2:=nuevo;
end;
procedure cargar_lista_isbns2(var lista_isbns2: t_lista_isbns; abb_isbns: t_abb_isbns);
begin
  if (abb_isbns<>nil) then
    begin
      cargar_lista_isbns2(lista_isbns2,abb_isbns^.hd);
      agregar_adelante_lista_isbns2(lista_isbns2,abb_isbns^.ele);
      cargar_lista_isbns2(lista_isbns2,abb_isbns^.hi);
    end;
  end;
end;
procedure imprimir_registro_isbn2(registro_isbn2: t_registro_isbn2);
begin
  textcolor(green); write('El ISBN es '); textcolor(red); writeln(registro_isbn2.isbn);
  textcolor(green); write('La cantidad total de veces que se prestó es '); textcolor(red);
writeln(registro_isbn2.prestamos);
end;
procedure imprimir1_lista_isbns(lista_isbns: t_lista_isbns);
begin
  while (lista_isbns<>nil) do
    begin
      imprimir_registro_isbn2(lista_isbns^.ele);
      writeln();
      lista_isbns:=lista_isbns^.sig;
    end;
  end;
end;
procedure imprimir2_lista_isbns(lista_isbns: t_lista_isbns);
begin
  if (lista_isbns<>nil) then

```

```

begin
    imprimir_registro_isbn2(lista_isbns^.ele);
    imprimir2_lista_isbns(lista_isbns^.sig);
end;
end;
procedure verificar_isbns(var isbn1, isbn2: int8);
var
    aux: int8;
begin
    if (isbn1>isbn2) then
    begin
        aux:=isbn1;
        isbn1:=isbn2;
        isbn2:=aux;
    end;
end;
function contar_isbns1(abb_prestamos: t_abb_prestamos; isbn1, isbn2: int8): int16;
begin
    if (abb_prestamos=nil) then
        contar_isbns1:=0
    else
        if (isbn1>abb_prestamos^.ele.isbn) then
            contar_isbns1:=contar_isbns1(abb_prestamos^.hd, isbn1, isbn2)
        else if (isbn2<abb_prestamos^.ele.isbn) then
            contar_isbns1:=contar_isbns1(abb_prestamos^.hi, isbn1, isbn2)
        else
            contar_isbns1:=contar_isbns1(abb_prestamos^.hi, isbn1, isbn2)+contar_isbns1(abb_prestamos^
            .hd, isbn1, isbn2)+1;
    end;
function contar_isbns2(abb_isbns: t_abb_isbns; isbn1, isbn2: int8): int16;
begin
    if (abb_isbns=nil) then
        contar_isbns2:=0
    else
        if (isbn1>abb_isbns^.ele.isbn) then
            contar_isbns2:=contar_isbns2(abb_isbns^.hd, isbn1, isbn2)
        else if (isbn2<abb_isbns^.ele.isbn) then
            contar_isbns2:=contar_isbns2(abb_isbns^.hi, isbn1, isbn2)
        else
            contar_isbns2:=contar_isbns2(abb_isbns^.hi, isbn1, isbn2)+contar_isbns2(abb_isbns^.hd, isbn
            1, isbn2)+contar_prestamos(abb_isbns^.ele.prestamos);
    end;
var
    lista_isbns1, lista_isbns2: t_lista_isbns;
    abb_prestamos: t_abb_prestamos;
    abb_isbns: t_abb_isbns;
    socio, isbn1, isbn2: int8;
begin
    randomize;
    abb_prestamos:=nil; abb_isbns:=nil;
    lista_isbns1:=nil; lista_isbns2:=nil;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_abbs(abb_prestamos, abb_isbns);
    if ((abb_prestamos<>nil) and (abb_isbns<>nil)) then
    begin
        writeln(); textcolor(red); writeln('ABB_PRESTAMOS:'); writeln();
        imprimir_abb_prestamos(abb_prestamos);
        writeln(); textcolor(red); writeln('ABB_ISBNS:'); writeln();
        imprimir_abb_isbns(abb_isbns);
        writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
        textcolor(green); write('El ISBN más grande es '); textcolor(red);
        writeln(buscar_mayor_isbn(abb_prestamos));
        writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
        textcolor(green); write('El ISBN más chico es '); textcolor(red);
        writeln(buscar_menor_isbn(abb_isbns));
        writeln(); textcolor(red); writeln('INCISO (d):'); writeln();

```

```
socio:=1+random(high(int8));
textcolor(green); write('La cantidad de préstamos en el abb_prestamos realizados al número
de socio '); textcolor(yellow); write(socio); textcolor(green); write(' es '); textcolor(red);
writeln(contar_abb_prestamos(abb_prestamos,socio));
writeln(); textcolor(red); writeln('INCISO (e):'); writeln();
socio:=1+random(high(int8));
textcolor(green); write('La cantidad de préstamos en el abb_isbns realizados al número de
socio '); textcolor(yellow); write(socio); textcolor(green); write(' es '); textcolor(red);
writeln(contar_abb_isbns(abb_isbns,socio));
writeln(); textcolor(red); writeln('INCISO (f):'); writeln();
cargar_lista_isbns1(lista_isbns1,abb_prestamos);
imprimir1_lista_isbns(lista_isbns1);
writeln(); textcolor(red); writeln('INCISO (g):'); writeln();
cargar_lista_isbns2(lista_isbns2,abb_isbns);
imprimir1_lista_isbns(lista_isbns2);
writeln(); textcolor(red); writeln('INCISO (h):'); writeln();
imprimir2_lista_isbns(lista_isbns1);
writeln();
imprimir2_lista_isbns(lista_isbns2);
writeln(); textcolor(red); writeln('INCISO (i):'); writeln();
isbn1:=1+random(high(int8)); isbn2:=1+random(high(int8));
verificar_isbns(isbn1,isbn2);
textcolor(green); write('La cantidad total de préstamos en el abb_prestamos cuyo ISBN se
encuentra entre '); textcolor(yellow); write(isbn1); textcolor(green); write(' y ');
textcolor(yellow); write(isbn2); textcolor(green); write(' (incluidos) es '); textcolor(red);
writeln(contar_isbns1(abb_prestamos,isbn1,isbn2));
writeln(); textcolor(red); writeln('INCISO (j):'); writeln();
textcolor(green); write('La cantidad total de préstamos en el abb_isbns cuyo ISBN se
encuentra entre '); textcolor(yellow); write(isbn1); textcolor(green); write(' y ');
textcolor(yellow); write(isbn2); textcolor(green); write(' (incluidos) es '); textcolor(red);
write(contar_isbns2(abb_isbns,isbn1,isbn2));
end;
end.
```

Ejercicio 3.

Una facultad nos ha encargado procesar la información de sus alumnos de la carrera XXX. Esta carrera tiene 30 materias. Implementar un programa con:

(a) Un módulo que lea la información de los finales rendidos por los alumnos y los almacene en dos estructuras de datos.

(i) Una estructura que, para cada alumno, se almacenen sólo código y nota de las materias aprobadas (4 a 10). De cada final rendido, se lee el código del alumno, el código de materia y la nota (valor entre 1 y 10). La lectura de los finales finaliza con nota -1. La estructura debe ser eficiente para buscar por código de alumno.

(ii) Otra estructura que almacene para cada materia, su código y todos los finales rendidos en esa materia (código de alumno y nota).

(b) Un módulo que reciba la estructura generada en (i) y un código de alumno y retorne los códigos y promedios de los alumnos cuyos códigos sean mayor al ingresado.

(c) Un módulo que reciba la estructura generada en (i), dos códigos de alumnos y un valor entero y retorne la cantidad de alumnos con cantidad de finales aprobados igual al valor ingresado para aquellos alumnos cuyos códigos están comprendidos entre los dos códigos de alumnos ingresados.

```
program TP4_E3;
{$codepage UTF8}
uses crt;
const
  materias_total=30;
  nota_corte=4;
  nota_ini=1; nota_fin=10;
  nota_salida=-1;
type
  t_materia=1..materias_total;
  t_nota=nota_salida..nota_fin;
  t_registro_final1=record
    codigo_alumno: int8;
    codigo_materia: t_materia;
    nota: t_nota;
  end;
  t_vector_notas=array[t_materia] of t_nota;
  t_registro_alumno1=record
    codigo_alumno: int8;
    notas: t_vector_notas;
  end;
  t_abb_alumnos1=^t_nodo_abb_alumnos1;
  t_nodo_abb_alumnos1=record
    ele: t_registro_alumno1;
    hi: t_abb_alumnos1;
    hd: t_abb_alumnos1;
  end;
  t_registro_final2=record
    codigo_alumno: int8;
    nota: t_nota;
  end;
  t_lista_finales=^t_nodo_finales;
  t_nodo_finales=record
    ele: t_registro_final2;
```

```

    sig: t_lista_finales;
end;
t_vector_finales=array[t_materia] of t_lista_finales;
t_registro_alumno2=record
    codigo_alumno: int8;
    promedio: real;
end;
t_abb_alumnos2:=t_nodo_abb_alumnos2;
t_nodo_abb_alumnos2=record
    ele: t_registro_alumno2;
    hi: t_abb_alumnos2;
    hd: t_abb_alumnos2;
end;
procedure inicializar_vector_finales(var vector_finales: t_vector_finales);
var
    i: t_materia;
begin
    for i:= 1 to materias_total do
        vector_finales[i]:=nil;
    end;
procedure leer_final(var registro_final1: t_registro_final1);
var
    i: int8;
begin
    i:=random(100);
    if (i=0) then
        registro_final1.nota:=nota_salida
    else
        registro_final1.nota:=nota_ini+random(nota_fin);
        if (registro_final1.nota<>nota_salida) then
            begin
                registro_final1.codigo_alumno:=1+random(high(int8));
                registro_final1.codigo_materia:=1+random(materias_total);
            end;
        end;
end;
procedure inicializar_vector_notas(var vector_notas: t_vector_notas);
var
    i: t_materia;
begin
    for i:= 1 to materias_total do
        vector_notas[i]:=0;
    end;
procedure cargar_registro_alumno1(var registro_alumno1: t_registro_alumno1; registro_final1:
t_registro_final1);
begin
    registro_alumno1.codigo_alumno:=registro_final1.codigo_alumno;
    inicializar_vector_notas(registro_alumno1.notas);
    if (registro_final1.nota>=nota_corte) then
        registro_alumno1.notas[registro_final1.codigo_materia]:=registro_final1.nota;
end;
procedure agregar_abb_alumnos1(var abb_alumnos1: t_abb_alumnos1; registro_final1:
t_registro_final1);
begin
    if (abb_alumnos1=nil) then
        begin
            new(abb_alumnos1);
            cargar_registro_alumno1(abb_alumnos1^.ele,registro_final1);
            abb_alumnos1^.hi:=nil;
            abb_alumnos1^.hd:=nil;
        end
    else
        if (registro_final1.codigo_alumno=abb_alumnos1^.ele.codigo_alumno) then
            begin
                if (registro_final1.nota>=nota_corte) then
                    abb_alumnos1^.ele.notas[registro_final1.codigo_materia]:=registro_final1.nota;
                end
            end
        end
    end
end

```

```

    else if (registro_final1.codigo_alumno<abb_alumnos1^.ele.codigo_alumno) then
        agregar_abb_alumnos1(abb_alumnos1^.hi,registro_final1)
    else
        agregar_abb_alumnos1(abb_alumnos1^.hd,registro_final1);
end;
procedure cargar_registro_final2(var registro_final2: t_registro_final2; registro_final1:
t_registro_final1);
begin
    registro_final2.codigo_alumno:=registro_final1.codigo_alumno;
    registro_final2.nota:=registro_final1.nota;
end;
procedure agregar_adelante_lista_finales(var lista_finales: t_lista_finales; registro_final1:
t_registro_final1);
var
    nuevo: t_lista_finales;
begin
    new(nuevo);
    cargar_registro_final2(nuevo^.ele,registro_final1);
    nuevo^.sig:=lista_finales;
    lista_finales:=nuevo;
end;
procedure cargar_vector_finales(var vector_finales: t_vector_finales; registro_final1:
t_registro_final1);
begin
    agregar_adelante_lista_finales(vector_finales[registro_final1.codigo_materia],registro_final
1);
end;
procedure cargar_estructuras(var abb_alumnos1: t_abb_alumnos1; var vector_finales:
t_vector_finales);
var
    registro_final1: t_registro_final1;
begin
    leer_final(registro_final1);
    while (registro_final1.nota<>nota_salida) do
        begin
            agregar_abb_alumnos1(abb_alumnos1,registro_final1);
            cargar_vector_finales(vector_finales,registro_final1);
            leer_final(registro_final1);
        end;
    end;
end;
procedure imprimir_vector_notas(vector_notas: t_vector_notas; codigo_alumno: int8);
var
    i: t_materia;
begin
    for i:= 1 to materias_total do
        begin
            if (vector_notas[i]>0) then
                begin
                    textcolor(green); write('La nota de la materia '); textcolor(yellow); write(i);
                    textcolor(green); write(' del código de alumno '); textcolor(yellow); write(codigo_alumno);
                    textcolor(green); write(' es '); textcolor(red); writeln(vector_notas[i]);
                end;
            end;
        end;
    end;
end;
procedure imprimir_registro_alumno1(registro_alumno1: t_registro_alumno1);
begin
    textcolor(green); write('El código de alumno del alumno es '); textcolor(red);
    writeln(registro_alumno1.codigo_alumno);
    imprimir_vector_notas(registro_alumno1.notas,registro_alumno1.codigo_alumno);
    writeln();
end;
procedure imprimir_abb_alumnos1(abb_alumnos1: t_abb_alumnos1);
begin
    if (abb_alumnos1<>nil) then
        begin
            imprimir_abb_alumnos1(abb_alumnos1^.hi);

```



```

    imprimir_registro_alumno1(abb_alumnos1^.ele);
    imprimir_abb_alumnos1(abb_alumnos1^.hd);
end;
end;
procedure imprimir_registro_final2(registro_final2: t_registro_final2; materia: t_materia;
final: int16);
begin
    textcolor(green); write('El código de alumno del final '); textcolor(yellow); write(final);
textcolor(green); write(' de la materia '); textcolor(yellow); write(materia);
textcolor(green); write(' es '); textcolor(red); writeln(registro_final2.codigo_alumno);
    textcolor(green); write('La nota del final '); textcolor(yellow); write(final);
textcolor(green); write(' de la materia '); textcolor(yellow); write(materia);
textcolor(green); write(' es '); textcolor(red); writeln(registro_final2.nota);
end;
procedure imprimir_lista_finales(lista_finales: t_lista_finales; materia: t_materia);
var
    i: int16;
begin
    i:=0;
    while (lista_finales<>nil) do
        begin
            i:=i+1;
            imprimir_registro_final2(lista_finales^.ele,materia,i);
            lista_finales:=lista_finales^.sig;
        end;
    end;
end;
procedure imprimir_vector_finales(vector_finales: t_vector_finales);
var
    i: t_materia;
begin
    for i:= 1 to materias_total do
        begin
            textcolor(green); write('Los finales rendidos de la materia '); textcolor(yellow);
write(i); textcolor(green); writeln(' son:');
            imprimir_lista_finales(vector_finales[i],i);
            writeln();
        end;
    end;
end;
function calcular_promedio(vector_notas: t_vector_notas): real;
var
    i: t_materia;
    notas_total, notas: int16;
begin
    notas_total:=0; notas:=0;
    for i:= 1 to materias_total do
        if (vector_notas[i]>=nota_corte) then
            begin
                notas_total:=notas_total+vector_notas[i];
                notas:=notas+1;
            end;
        if (notas>0) then
            calcular_promedio:=notas_total/notas
        else
            calcular_promedio:=notas_total;
        end;
    end;
end;
procedure cargar_registro_alumno2(var registro_alumno2: t_registro_alumno2; registro_alumno1:
t_registro_alumno1);
begin
    registro_alumno2.codigo_alumno:=registro_alumno1.codigo_alumno;
    registro_alumno2.promedio:=calcular_promedio(registro_alumno1.notas);
end;
procedure agregar_abb_alumnos2(var abb_alumnos2: t_abb_alumnos2; registro_alumno1:
t_registro_alumno1);
begin
    if (abb_alumnos2=nil) then
        begin

```

```

new(abb_alumnos2);
cargar_registro_alumno2(abb_alumnos2^.ele,registro_alumno1);
abb_alumnos2^.hi:=nil;
abb_alumnos2^.hd:=nil;
end
else
  if (registro_alumno1.codigo_alumno<=abb_alumnos2^.ele.codigo_alumno) then
    agregar_abb_alumnos2(abb_alumnos2^.hi,registro_alumno1)
  else
    agregar_abb_alumnos2(abb_alumnos2^.hd,registro_alumno1);
end;
procedure cargar_abb_alumnos2(var abb_alumnos2: t_abb_alumnos2; abb_alumnos1: t_abb_alumnos1;
codigo: int8);
begin
  if (abb_alumnos1<>nil) then
    begin
      if (abb_alumnos1^.ele.codigo_alumno>codigo) then
        begin
          cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.hi,codigo);
          agregar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.ele);
          cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.hd,codigo);
        end
      else
        cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1^.hd,codigo);
      end;
    end;
end;
procedure imprimir_registro_alumno2(registro_alumno2: t_registro_alumno2);
begin
  textcolor(green); write('El código de alumno del alumno es '); textcolor(red);
writeln(registro_alumno2.codigo_alumno);
  textcolor(green); write('El promedio del alumno es '); textcolor(red);
writeln(registro_alumno2.promedio:0:2);
  writeln();
end;
procedure imprimir_abb_alumnos2(abb_alumnos2: t_abb_alumnos2);
begin
  if (abb_alumnos2<>nil) then
    begin
      imprimir_abb_alumnos2(abb_alumnos2^.hi);
      imprimir_registro_alumno2(abb_alumnos2^.ele);
      imprimir_abb_alumnos2(abb_alumnos2^.hd);
    end;
end;
procedure verificar_codigos(var codigo1, codigo2: int8);
var
  aux: int8;
begin
  if (codigo1>codigo2) then
    begin
      aux:=codigo1;
      codigo1:=codigo2;
      codigo2:=aux;
    end;
end;
function contar_notas(vector_notas: t_vector_notas; finales: t_materia): int8;
var
  i: t_materia;
  notas: int8;
begin
  notas:=0;
  for i:= 1 to materias_total do
    if (vector_notas[i]>=nota_corte) then
      notas:=notas+1;
    if (notas=finales) then
      contar_notas:=1
    else

```

```

    contar_notas:=0;
end;
function contar_alumnos(abb_alumnos1: t_abb_alumnos1; codigo1, codigo2: int16; finales:
t_materia): int16;
begin
    if (abb_alumnos1=nil) then
        contar_alumnos:=0
    else
        if (codigo1>=abb_alumnos1^.ele.codigo_alumno) then
            contar_alumnos:=contar_alumnos(abb_alumnos1^.hd,codigo1,codigo2,finales)
        else if (codigo2<=abb_alumnos1^.ele.codigo_alumno) then
            contar_alumnos:=contar_alumnos(abb_alumnos1^.hi,codigo1,codigo2,finales)
        else
            contar_alumnos:=contar_alumnos(abb_alumnos1^.hi,codigo1,codigo2,finales)+contar_alumnos(
abb_alumnos1^.hd,codigo1,codigo2,finales)+contar_notas(abb_alumnos1^.ele.notas,finales);
        end;
    end;
var
    vector_finales: t_vector_finales;
    abb_alumnos1: t_abb_alumnos1;
    abb_alumnos2: t_abb_alumnos2;
    finales: t_materia;
    codigo, codigo1, codigo2: int8;
begin
    randomize;
    abb_alumnos1:=nil; inicializar_vector_finales(vector_finales);
    abb_alumnos2:=nil;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_estructuras(abb_alumnos1,vector_finales);
    if (abb_alumnos1<>nil) then
        begin
            writeln(); textcolor(red); writeln('ABB_ALUMNOS1:'); writeln();
            imprimir_abb_alumnos1(abb_alumnos1);
            writeln(); textcolor(red); writeln('VECTOR_FINALES:'); writeln();
            imprimir_vector_finales(vector_finales);
            writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
            codigo:=1+random(high(int8));
            cargar_abb_alumnos2(abb_alumnos2,abb_alumnos1,codigo);
            if (abb_alumnos2<>nil) then
                imprimir_abb_alumnos2(abb_alumnos2);
                writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
                codigo1:=1+random(high(int8)); codigo2:=1+random(high(int8)); finales:=2;
                verificar_codigos(codigo1,codigo2);
                textcolor(green); write('La cantidad de alumnos en el abb cuyo código de alumno se
encuentra entre '); textcolor(yellow); write(codigo1); textcolor(green); write(' y ');
textcolor(yellow); write(codigo2); textcolor(green); write(' y tienen '); textcolor(yellow);
write(finales); textcolor(green); write(' finales aprobados es '); textcolor(red);
write(contar_alumnos(abb_alumnos1,codigo1,codigo2,finales));
            end;
        end;
    end.

```

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_total=300;
  codigo_salida=-1;
type
  t_oficina=1..oficinas_total;
  t_registro_oficina=record
    codigo: int16;
    dni: int32;
    expensa: real;
  end;
  t_vector_oficinas=array[t_oficina] of t_registro_oficina;
procedure leer_oficina(var registro_oficina: t_registro_oficina);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_oficina.codigo:=codigo_salida
  else
    registro_oficina.codigo:=random(high(int16));
    if (registro_oficina.codigo<>codigo_salida) then
      begin
        registro_oficina.dni:=10000000+random(40000001);
        registro_oficina.expensa:=1+random(100);
      end;
    end;
end;
procedure cargar_vector_oficinas(var vector_oficinas: t_vector_oficinas; var oficinas: int16);
var
```

```
registro_oficina: t_registro_oficina;
begin
  leer_oficina(registro_oficina);
  while ((registro_oficina.codigo<>codigo_salida) and (oficinas<oficinas_total)) do
    begin
      oficinas:=oficinas+1;
      vector_oficinas[oficinas]:=registro_oficina;
      leer_oficina(registro_oficina);
    end;
  end;
end;
procedure imprimir_registro_oficina(registro_oficina: t_registro_oficina; oficina: t_oficina);
begin
  textcolor(green); write('El código de identificación de la oficina '); textcolor(yellow);
  write(oficina); textcolor(green); write(' es '); textcolor(red);
  writeln(registro_oficina.codigo);
  textcolor(green); write('El DNI del propietario de la oficina '); textcolor(yellow);
  write(oficina); textcolor(green); write(' es '); textcolor(red);
  writeln(registro_oficina.dni);
  textcolor(green); write('El valor de la expensa de la oficina '); textcolor(yellow);
  write(oficina); textcolor(green); write(' es $'); textcolor(red);
  writeln(registro_oficina.expensa:0:2);
end;
procedure imprimir_vector_oficinas(vector_oficinas: t_vector_oficinas; oficinas: int16);
var
  i: t_oficina;
begin
  for i:= 1 to oficinas do
    begin
      textcolor(green); write('La información de la oficina '); textcolor(yellow); write(i);
      textcolor(green); writeln(' es:');
      imprimir_registro_oficina(vector_oficinas[i],i);
      writeln();
    end;
  end;
end;
procedure ordenar_vector_oficinas(var vector_oficinas: t_vector_oficinas; oficinas: int16);
var
  item: t_registro_oficina;
  i, j, k: t_oficina;
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];
        vector_oficinas[k]:=vector_oficinas[i];
        vector_oficinas[i]:=item;
      end;
    end;
  end;
end;
function buscar_vector_oficinas(vector_oficinas: t_vector_oficinas; codigo, pri, ult: int16):
int16;
var
  medio: int8;
begin
  if (pri<=ult) then
    begin
      medio:=(pri+ult) div 2;
      if (codigo=vector_oficinas[medio].codigo) then
        buscar_vector_oficinas:=medio
      else if (codigo<vector_oficinas[medio].codigo) then
        buscar_vector_oficinas:=buscar_vector_oficinas(vector_oficinas,codigo,pri,medio-1)
      else
        buscar_vector_oficinas:=buscar_vector_oficinas(vector_oficinas,codigo,medio+1,ult)
      end
    end
  else
    buscar_vector_oficinas:=0;
```

```
        buscar_vector_oficinas:=0;
end;
function sumar_vector_oficinas(vector_oficinas: t_vector_oficinas; oficinas: int16): real;
begin
    if (oficinas=1) then
        sumar_vector_oficinas:=vector_oficinas[oficinas].expensa
    else
        sumar_vector_oficinas:=sumar_vector_oficinas(vector_oficinas,oficinas-
1)+vector_oficinas[oficinas].expensa;
    end;
var
    vector_oficinas: t_vector_oficinas;
    oficinas, codigo, pri, ult, pos: int16;
begin
    randomize;
    oficinas:=0;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_vector_oficinas(vector_oficinas,oficinas);
    if (oficinas>0) then
    begin
        imprimir_vector_oficinas(vector_oficinas,oficinas);
        writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
        ordenar_vector_oficinas(vector_oficinas,oficinas);
        imprimir_vector_oficinas(vector_oficinas,oficinas);
        writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
        codigo:=1+random(high(int16));
        pri:=1; ult:=oficinas;
        pos:=buscar_vector_oficinas(vector_oficinas,codigo,pri,ult);
        if (pos<>0) then
        begin
            textcolor(green); write('El código de identificación de oficina '); textcolor(yellow);
            write(codigo); textcolor(green); write(' se encontró en el vector, en la posición ');
            textcolor(red); writeln(pos);
            textcolor(green); write('El DNI del propietario de la oficina con código de
identificación '); textcolor(yellow); write(codigo); textcolor(green); write(' es ');
            textcolor(red); writeln(vector_oficinas[pos].dni);
        end
        else
        begin
            textcolor(green); write('El código de identificación de oficina '); textcolor(yellow);
            write(codigo); textcolor(green); writeln(' no se encontró en el vector');
        end;
        writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
        textcolor(green); write('El monto total de las expensas es $'); textcolor(red);
        write(sumar_vector_oficinas(vector_oficinas,oficinas):0:2);
    end;
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_ini=2010; anio_fin=2018;
  marca_salida='MMM';
type
  t_anio=anio_ini..anio_fin;
  t_registro_auto1=record
    patente: string;
    anio: t_anio;
    marca: string;
    modelo: string;
  end;
  t_abb_patentes=^t_nodo_abb_patentes;
  t_nodo_abb_patentes=record
    ele: t_registro_auto1;
    hi: t_abb_patentes;
    hd: t_abb_patentes;
  end;
  t_registro_auto2=record
    patente: string;
    anio: t_anio;
    modelo: string;
  end;
  t_lista_autos1=^t_nodo_autos1;
  t_nodo_autos1=record
    ele: t_registro_auto2;
    sig: t_lista_autos1;
```

```
end;
t_registro_marca=record
  marca: string;
  autos: t_lista_autos1;
end;
t_abb_marcas=^t_nodo_abb_marcas;
t_nodo_abb_marcas=record
  ele: t_registro_marca;
  hi: t_abb_marcas;
  hd: t_abb_marcas;
end;
t_registro_auto3=record
  patente: string;
  marca: string;
  modelo: string;
end;
t_lista_autos2=^t_nodo_autos2;
t_nodo_autos2=record
  ele: t_registro_auto3;
  sig: t_lista_autos2;
end;
t_vector_autos=array[t_anio] of t_lista_autos2;
procedure inicializar_vector_autos(var vector_autos: t_vector_autos);
var
  i: t_anio;
begin
  for i:= anio_ini to anio_fin do
    vector_autos[i]:=nil;
  end;
end;
function random_string(length: int8): string;
var
  i: int8;
  string_aux: string;
begin
  string_aux:='';
  for i:= 1 to length do
    string_aux:=string_aux+chr(ord('A')+random(26));
  end;
  random_string:=string_aux;
end;
procedure leer_auto(var registro_auto1: t_registro_auto1);
var
  i: int16;
begin
  i:=random(100);
  if (i=0) then
    registro_auto1.marca:=marca_salida
  else
    registro_auto1.marca:='Marca '+random_string(1);
    if (registro_auto1.marca<>marca_salida) then
      begin
        registro_auto1.patente:=random_string(2);
        registro_auto1.anio:=anio_ini+random(anio_fin-anio_ini+1);
        registro_auto1.modelo:='Modelo '+random_string(2);
      end;
    end;
end;
procedure agregar_abb_patentes(var abb_patentes: t_abb_patentes; registro_auto1:
t_registro_auto1);
begin
  if (abb_patentes=nil) then
    begin
      new(abb_patentes);
      abb_patentes^.ele:=registro_auto1;
      abb_patentes^.hi:=nil;
      abb_patentes^.hd:=nil;
    end
  else
```



```
    if (registro_auto1.patente<=abb_patentes^.ele.patente) then
        agregar_abb_patentes(abb_patentes^.hi,registro_auto1)
    else
        agregar_abb_patentes(abb_patentes^.hd,registro_auto1);
end;
procedure cargar_registro_auto2(var registro_auto2: t_registro_auto2; registro_auto1:
t_registro_auto1);
begin
    registro_auto2.patente:=registro_auto1.patente;
    registro_auto2.anio:=registro_auto1.anio;
    registro_auto2.modelo:=registro_auto1.modelo;
end;
procedure agregar_adelante_lista_autos1(var lista_autos1: t_lista_autos1; registro_auto1:
t_registro_auto1);
var
    nuevo: t_lista_autos1;
begin
    new(nuevo);
    cargar_registro_auto2(nuevo^.ele,registro_auto1);
    nuevo^.sig:=lista_autos1;
    lista_autos1:=nuevo;
end;
procedure cargar_registro_marca(var registro_marca: t_registro_marca; registro_auto1:
t_registro_auto1);
begin
    registro_marca.marca:=registro_auto1.marca;
    registro_marca.autos:=nil;
    agregar_adelante_lista_autos1(registro_marca.autos,registro_auto1);
end;
procedure agregar_abb_marcas(var abb_marcas: t_abb_marcas; registro_auto1: t_registro_auto1);
begin
    if (abb_marcas=nil) then
        begin
            new(abb_marcas);
            cargar_registro_marca(abb_marcas^.ele,registro_auto1);
            abb_marcas^.hi:=nil;
            abb_marcas^.hd:=nil;
        end
    else
        if (registro_auto1.marca=abb_marcas^.ele.marca) then
            agregar_adelante_lista_autos1(abb_marcas^.ele.autos,registro_auto1)
        else if (registro_auto1.marca<abb_marcas^.ele.marca) then
            agregar_abb_marcas(abb_marcas^.hi,registro_auto1)
        else
            agregar_abb_marcas(abb_marcas^.hd,registro_auto1);
        end
    end;
end;
procedure cargar_abbs(var abb_patentes: t_abb_patentes; var abb_marcas: t_abb_marcas);
var
    registro_auto1: t_registro_auto1;
begin
    leer_auto(registro_auto1);
    while (registro_auto1.marca<>marca_salida) do
        begin
            agregar_abb_patentes(abb_patentes,registro_auto1);
            agregar_abb_marcas(abb_marcas,registro_auto1);
            leer_auto(registro_auto1);
        end
    end;
end;
procedure imprimir_registro_auto1(registro_auto1: t_registro_auto1);
begin
    textcolor(green); write('La patente del auto es '); textcolor(red);
writeln(registro_auto1.patente);
    textcolor(green); write('El año de fabricación del auto es '); textcolor(red);
writeln(registro_auto1.anio);
    textcolor(green); write('La marca del auto es '); textcolor(red);
writeln(registro_auto1.marca);
```

```

    textcolor(green); write('El modelo del auto es '); textcolor(red);
writeln(registro_auto1.modelo);
    writeln();
end;
procedure imprimir_abb_patentes(abb_patentes: t_abb_patentes);
begin
    if (abb_patentes<>nil) then
    begin
        imprimir_abb_patentes(abb_patentes^.hi);
        imprimir_registro_auto1(abb_patentes^.ele);
        imprimir_abb_patentes(abb_patentes^.hd);
    end;
end;
procedure imprimir_registro_auto2(registro_auto2: t_registro_auto2; marca: string; auto:
int16);
begin
    textcolor(green); write('La patente del auto '); textcolor(yellow); write(auto);
textcolor(green); write(' de la marca '); textcolor(yellow); write(marca); textcolor(green);
write(' es '); textcolor(red); writeln(registro_auto2.patente);
    textcolor(green); write('El año de fabricación del auto '); textcolor(yellow); write(auto);
textcolor(green); write(' de la marca '); textcolor(yellow); write(marca); textcolor(green);
write(' es '); textcolor(red); writeln(registro_auto2.anio);
    textcolor(green); write('El modelo del auto '); textcolor(yellow); write(auto);
textcolor(green); write(' de la marca '); textcolor(yellow); write(marca); textcolor(green);
write(' es '); textcolor(red); writeln(registro_auto2.modelo);
end;
procedure imprimir_lista_autos1(lista_autos1: t_lista_autos1; marca: string);
var
    i: int16;
begin
    i:=0;
    while (lista_autos1<>nil) do
    begin
        i:=i+1;
        imprimir_registro_auto2(lista_autos1^.ele,marca,i);
        lista_autos1:=lista_autos1^.sig;
    end;
end;
procedure imprimir_registro_marca(registro_marca: t_registro_marca);
begin
    textcolor(green); write('La marca del auto es '); textcolor(red);
writeln(registro_marca.marca);
    imprimir_lista_autos1(registro_marca.autos,registro_marca.marca);
    writeln();
end;
procedure imprimir_abb_marcas(abb_marcas: t_abb_marcas);
begin
    if (abb_marcas<>nil) then
    begin
        imprimir_abb_marcas(abb_marcas^.hi);
        imprimir_registro_marca(abb_marcas^.ele);
        imprimir_abb_marcas(abb_marcas^.hd);
    end;
end;
function contar_abb_patentes(abb_patentes: t_abb_patentes; marca: string): int8;
begin
    if (abb_patentes=nil) then
        contar_abb_patentes:=0
    else
        if (marca=abb_patentes^.ele.marca) then
            contar_abb_patentes:=contar_abb_patentes(abb_patentes^.hi,marca)+contar_abb_patentes(abb
_patentes^.hd,marca)+1
        else
            contar_abb_patentes:=contar_abb_patentes(abb_patentes^.hi,marca)+contar_abb_patentes(abb
_patentes^.hd,marca);
        end;
    end;
end;

```

```

function contar_autos(lista_autos1: t_lista_autos1): int8;
var
  autos: int8;
begin
  autos:=0;
  while (lista_autos1<>nil) do
    begin
      autos:=autos+1;
      lista_autos1:=lista_autos1^.sig;
    end;
  contar_autos:=autos;
end;
function contar_abb_marcas(abb_marcas: t_abb_marcas; marca: string): int8;
begin
  if (abb_marcas=nil) then
    contar_abb_marcas:=0
  else
    if (marca=abb_marcas^.ele.marca) then
      contar_abb_marcas:=contar_autos(abb_marcas^.ele.autos)
    else if (marca<abb_marcas^.ele.marca) then
      contar_abb_marcas:=contar_abb_marcas(abb_marcas^.hi,marca)
    else
      contar_abb_marcas:=contar_abb_marcas(abb_marcas^.hd,marca)
    end;
end;
procedure cargar_registro_auto3(var registro_auto3: t_registro_auto3; registro_auto1:
t_registro_auto1);
begin
  registro_auto3.patente:=registro_auto1.patente;
  registro_auto3.marca:=registro_auto1.marca;
  registro_auto3.modelo:=registro_auto1.modelo;
end;
procedure agregar_adelante_lista_autos2(var lista_autos2: t_lista_autos2; registro_auto1:
t_registro_auto1);
var
  nuevo: t_lista_autos2;
begin
  new(nuevo);
  cargar_registro_auto3(nuevo^.ele,registro_auto1);
  nuevo^.sig:=lista_autos2;
  lista_autos2:=nuevo;
end;
procedure cargar_vector_autos(var vector_autos: t_vector_autos; abb_patentes: t_abb_patentes);
begin
  if (abb_patentes<>nil) then
    begin
      cargar_vector_autos(vector_autos,abb_patentes^.hi);
      agregar_adelante_lista_autos2(vector_autos[abb_patentes^.ele.anio],abb_patentes^.ele);
      cargar_vector_autos(vector_autos,abb_patentes^.hd);
    end
  end;
end;
procedure imprimir_registro_auto3(registro_auto3: t_registro_auto3; anio: t_anio; auto:
int16);
begin
  textcolor(green); write('La patente del auto '); textcolor(yellow); write(auto);
textcolor(green); write(' del año de fabricación '); textcolor(yellow); write(anio);
textcolor(green); write(' es '); textcolor(red); writeln(registro_auto3.patente);
  textcolor(green); write('La marca el auto '); textcolor(yellow); write(auto);
textcolor(green); write(' del año de fabricación '); textcolor(yellow); write(anio);
textcolor(green); write(' es '); textcolor(red); writeln(registro_auto3.marca);
  textcolor(green); write('El modelo del auto '); textcolor(yellow); write(auto);
textcolor(green); write(' del año de fabricación '); textcolor(yellow); write(anio);
textcolor(green); write(' es '); textcolor(red); writeln(registro_auto3.modelo);
end;
procedure imprimir_lista_autos2(lista_autos2: t_lista_autos2; anio: t_anio);
var
  i: int16;

```

```

begin
  i:=0;
  while (lista_autos2<>nil) do
    begin
      i:=i+1;
      imprimir_registro_auto3(lista_autos2^.ele,anio,i);
      lista_autos2:=lista_autos2^.sig;
    end;
  end;
procedure imprimir_vector_autos(vector_autos: t_vector_autos);
var
  i: t_anio;
begin
  for i:= anio_ini to anio_fin do
    begin
      textcolor(green); write('La información de los autos del año de fabricación ');
      textcolor(yellow); write(i); textcolor(green); writeln(' es:');
      imprimir_lista_autos2(vector_autos[i],i);
      writeln();
    end;
  end;
function buscar_abb_patentes(abb_patentes: t_abb_patentes; patente: string): string;
begin
  if (abb_patentes=nil) then
    buscar_abb_patentes:='No existe la patente'
  else
    if (patente=abb_patentes^.ele.patente) then
      buscar_abb_patentes:=abb_patentes^.ele.modelo
    else if (patente<abb_patentes^.ele.patente) then
      buscar_abb_patentes:=buscar_abb_patentes(abb_patentes^.hi,patente)
    else
      buscar_abb_patentes:=buscar_abb_patentes(abb_patentes^.hd,patente);
  end;
function buscar_patente(lista_autos1: t_lista_autos1; patente: string): string;
begin
  while ((lista_autos1<>nil) and (lista_autos1^.ele.patente<>patente)) do
    lista_autos1:=lista_autos1^.sig;
  if (lista_autos1<>nil) then
    buscar_patente:=lista_autos1^.ele.modelo
  else
    buscar_patente:='No existe la patente';
  end;
function buscar_abb_marcas(abb_marcas: t_abb_marcas; patente: string): string;
var
  modelo: string;
begin
  if (abb_marcas=nil) then
    buscar_abb_marcas:='No existe la patente'
  else
    begin
      modelo:=buscar_patente(abb_marcas^.ele.autos,patente);
      if (modelo='No existe la patente') then
        modelo:=buscar_abb_marcas(abb_marcas^.hi,patente);
      if (modelo='No existe la patente') then
        modelo:=buscar_abb_marcas(abb_marcas^.hd,patente);
      buscar_abb_marcas:=modelo;
    end;
  end;
var
  vector_autos: t_vector_autos;
  abb_patentes: t_abb_patentes;
  abb_marcas: t_abb_marcas;
  marca, patente: string;
begin
  randomize;
  abb_patentes:=nil; abb_marcas:=nil;

```

```
inicializar_vector_autos(vector_autos);
writeln(); textcolor(red); writeln('INCISO (a)'); writeln();
cargar_abbs(abb_patentes,abb_marcas);
if ((abb_patentes<>nil) and (abb_marcas<>nil)) then
begin
    writeln(); textcolor(red); writeln('ABB_PATENTES:'); writeln();
    imprimir_abb_patentes(abb_patentes);
    writeln(); textcolor(red); writeln('ABB_MARCAS:'); writeln();
    imprimir_abb_marcas(abb_marcas);
    writeln(); textcolor(red); writeln('INCISO (b)'); writeln();
    marca:='Marca '+random_string(1);
    textcolor(green); write('La cantidad de autos en el abb_patentes de la marca ');
    textcolor(yellow); write(marca); textcolor(green); write(' es '); textcolor(red);
    writeln(contar_abb_patentes(abb_patentes,marca));
    writeln(); textcolor(red); writeln('INCISO (c)'); writeln();
    textcolor(green); write('La cantidad de autos en el abb_marcas de la marca ');
    textcolor(yellow); write(marca); textcolor(green); write(' es '); textcolor(red);
    writeln(contar_abb_marcas(abb_marcas,marca));
    writeln(); textcolor(red); writeln('INCISO (d)'); writeln();
    cargar_vector_autos(vector_autos,abb_patentes);
    imprimir_vector_autos(vector_autos);
    writeln(); textcolor(red); writeln('INCISO (e)'); writeln();
    patente:=random_string(2);
    textcolor(green); write('El modelo del auto de la patente '); textcolor(yellow);
    write(patente); textcolor(green); write(' es '); textcolor(red);
    writeln(buscar_abb_patentes(abb_patentes,patente));
    writeln(); textcolor(red); writeln('INCISO (f)'); writeln();
    textcolor(green); write('El modelo del auto de la patente '); textcolor(yellow);
    write(patente); textcolor(green); write(' es '); textcolor(red);
    write(buscar_abb_marcas(abb_marcas,patente));
end;
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_ini=1; rubro_fin=10;
    codigo_salida=-1;
type
    t_rubro=rubro_ini..rubro_fin;
    t_registro_producto1=record
        codigo: int16;
        rubro: t_rubro;
        stock: int16;
        precio: real;
    end;
    t_registro_producto2=record
        codigo: int16;
        stock: int16;
        precio: real;
    end;
    t_abb_productos=^t_nodo_abb_productos;
    t_nodo_abb_productos=record
        ele: t_registro_producto2;
        hi: t_abb_productos;
        hd: t_abb_productos;
    end;
    t_vector_abbs=array[t_rubro] of t_abb_productos;
    t_registro_producto3=record
        codigo: int16;
        stock: int16;
    end;
    t_vector_productos=array[t_rubro] of t_registro_producto3;
    t_vector_cantidades=array[t_rubro] of int16;
procedure inicializar_vector_abbs(var vector_abbs: t_vector_abbs);
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        vector_abbs[i]:=nil;
    end;
procedure inicializar_vector_productos(var vector_productos: t_vector_productos);

```

```
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        begin
            vector_productos[i].codigo:=codigo_salida;
            vector_productos[i].stock:=0;
        end;
    end;
procedure inicializar_vector_cantidades(var vector_cantidades: t_vector_cantidades);
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        vector_cantidades[i]:=0;
    end;
procedure leer_producto(var registro_producto1: t_registro_producto1);
var
    i: int8;
begin
    i:=random(100);
    if (i=0) then
        registro_producto1.codigo:=codigo_salida
    else
        registro_producto1.codigo:=1+random(high(int16));
    if (registro_producto1.codigo<>codigo_salida) then
        begin
            registro_producto1.rubro:=rubro_ini+random(rubro_fin);
            registro_producto1.stock:=1+random(high(int16));
            registro_producto1.precio:=1+random(100);
        end;
    end;
end;
procedure cargar_registro_producto2(var registro_producto2: t_registro_producto2;
registro_producto1: t_registro_producto1);
begin
    registro_producto2.codigo:=registro_producto1.codigo;
    registro_producto2.stock:=registro_producto1.stock;
    registro_producto2.precio:=registro_producto1.precio;
end;
procedure agregar_abb_productos(var abb_productos: t_abb_productos; registro_producto1:
t_registro_producto1);
begin
    if (abb_productos=nil) then
        begin
            new(abb_productos);
            cargar_registro_producto2(abb_productos^.ele,registro_producto1);
            abb_productos^.hi:=nil;
            abb_productos^.hd:=nil;
        end
    else
        if (registro_producto1.codigo<=abb_productos^.ele.codigo) then
            agregar_abb_productos(abb_productos^.hi,registro_producto1)
        else
            agregar_abb_productos(abb_productos^.hd,registro_producto1);
        end;
end;
procedure cargar_vector_abbs(var vector_abbs: t_vector_abbs);
var
    registro_producto1: t_registro_producto1;
begin
    leer_producto(registro_producto1);
    while (registro_producto1.codigo<>codigo_salida) do
        begin
            agregar_abb_productos(vector_abbs[registro_producto1.rubro],registro_producto1);
            leer_producto(registro_producto1);
        end;
    end;
end;
```

```

procedure imprimir_registro_producto2(registro_producto2: t_registro_producto2; rubro:
t_rubro);
begin
    textcolor(green); write('El código de producto del producto del rubro '); textcolor(yellow);
write(rubro); textcolor(green); write(' es '); textcolor(red);
writeln(registro_producto2.codigo);
    textcolor(green); write('El stock del producto del rubro '); textcolor(yellow);
write(rubro); textcolor(green); write(' es '); textcolor(red);
writeln(registro_producto2.stock);
    textcolor(green); write('El precio del producto del rubro '); textcolor(yellow);
write(rubro); textcolor(green); write(' es '); textcolor(red);
writeln(registro_producto2.precio:0:2);
end;
procedure imprimir_abb_productos(abb_productos: t_abb_productos; rubro: t_rubro);
begin
    if (abb_productos<>nil) then
        begin
            imprimir_abb_productos(abb_productos^.hi,rubro);
            imprimir_registro_producto2(abb_productos^.ele,rubro);
            imprimir_abb_productos(abb_productos^.hd,rubro);
        end;
    end;
procedure imprimir_vector_abbs(vector_abbs: t_vector_abbs);
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        begin
            textcolor(green); write('La información de los productos del rubro '); textcolor(yellow);
write(i); textcolor(green); writeln(' es:');
            imprimir_abb_productos(vector_abbs[i],i);
            writeln();
        end;
    end;
function buscar_abb_productos(abb_productos: t_abb_productos; codigo: int16): boolean;
begin
    if (abb_productos=nil) then
        buscar_abb_productos:=false
    else
        if (codigo=abb_productos^.ele.codigo) then
            buscar_abb_productos:=true
        else if (codigo<abb_productos^.ele.codigo) then
            buscar_abb_productos:=buscar_abb_productos(abb_productos^.hi,codigo)
        else
            buscar_abb_productos:=buscar_abb_productos(abb_productos^.hd,codigo);
    end;
procedure cargar_registro_producto3(var registro_producto3: t_registro_producto3;
abb_productos: t_abb_productos);
begin
    if (abb_productos^.hd=nil) then
        begin
            registro_producto3.codigo:=abb_productos^.ele.codigo;
            registro_producto3.stock:=abb_productos^.ele.stock;
        end
    else
        cargar_registro_producto3(registro_producto3,abb_productos^.hd);
end;
procedure cargar_vector_productos(var vector_productos: t_vector_productos; vector_abbs:
t_vector_abbs);
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        if (vector_abbs[i]<>nil) then
            cargar_registro_producto3(vector_productos[i],vector_abbs[i]);
    end;
end;

```



```

procedure imprimir_registro_producto3(registro_producto3: t_registro_producto3; rubro:
t_rubro);
begin
    textcolor(green); write('El mayor código de producto del rubro '); textcolor(yellow);
write(rubro); textcolor(green); write(' es '); textcolor(red);
writeln(registro_producto3.codigo);
    textcolor(green); write('El stock del mayor código de producto del rubro ');
textcolor(yellow); write(rubro); textcolor(green); write(' es '); textcolor(red);
writeln(registro_producto3.stock);
end;
procedure imprimir_vector_productos(vector_productos: t_vector_productos);
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        begin
            imprimir_registro_producto3(vector_productos[i],i);
            writeln();
        end;
    end;
procedure verificar_codigos(var codigo1, codigo2: int16);
var
    aux: int16;
begin
    if (codigo1>codigo2) then
        begin
            aux:=codigo1;
            codigo1:=codigo2;
            codigo2:=aux;
        end;
    end;
function contar_productos(abb_productos: t_abb_productos; codigo1, codigo2: int16): int16;
begin
    if (abb_productos=nil) then
        contar_productos:=0
    else
        if (codigo1>=abb_productos^.ele.codigo) then
            contar_productos:=contar_productos(abb_productos^.hd,codigo1,codigo2)
        else if (codigo2<=abb_productos^.ele.codigo) then
            contar_productos:=contar_productos(abb_productos^.hi,codigo1,codigo2)
        else
            contar_productos:=contar_productos(abb_productos^.hi,codigo1,codigo2)+contar_productos(a
bb_productos^.hd,codigo1,codigo2)+1;
    end;
procedure cargar_vector_cantidades(var vector_cantidades: t_vector_cantidades; vector_abbs:
t_vector_abbs; codigo1, codigo2: int16);
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        vector_cantidades[i]:=contar_productos(vector_abbs[i],codigo1,codigo2);
    end;
procedure imprimir_vector_cantidades(vector_cantidades: t_vector_cantidades; codigo1, codigo2:
int16);
var
    i: t_rubro;
begin
    for i:= rubro_ini to rubro_fin do
        begin
            textcolor(green); write('La cantidad de productos del rubro '); textcolor(yellow);
write(i); textcolor(green); write(' (cuyo código de producto se encuentra entre ');
textcolor(yellow); write(codigo1); textcolor(green); write(' y '); textcolor(yellow);
write(codigo2); textcolor(green); write(') es '); textcolor(red);
writeln(vector_cantidades[i]);
        end;
    end;
end;

```

```
var
  vector_abbs: t_vector_abbs;
  vector_productos: t_vector_productos;
  vector_cantidades: t_vector_cantidades;
  rubro: t_rubro;
  codigo, codigo1, codigo2: int16;
begin
  randomize;
  inicializar_vector_abbs(vector_abbs);
  inicializar_vector_productos(vector_productos);
  inicializar_vector_cantidades(vector_cantidades);
  writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
  cargar_vector_abbs(vector_abbs);
  imprimir_vector_abbs(vector_abbs);
  writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
  rubro:=rubro_ini+random(rubro_fin); codigo:=1+random(high(int16));
  textcolor(green); write('¿El código '); textcolor(yellow); write(codigo); textcolor(green);
  write(' se encuentra en el abb del rubro '); textcolor(yellow); write(rubro);
  textcolor(green); write('?: '); textcolor(red);
  writeln(buscar_abb_productos(vector_abbs[rubro],codigo));
  writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
  cargar_vector_productos(vector_productos,vector_abbs);
  imprimir_vector_productos(vector_productos);
  writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
  codigo1:=1+random(high(int16)); codigo2:=1+random(high(int16));
  verificar_codigos(codigo1,codigo2);
  cargar_vector_cantidades(vector_cantidades,vector_abbs,codigo1,codigo2);
  imprimir_vector_cantidades(vector_cantidades,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
  anio_ini=2000; anio_fin=2023;
  codigo_salida=-1;
type
  t_anio=anio_ini..anio_fin;
  t_registro_reclamo1=record
    codigo: int16;
    dni: int8;
    anio: t_anio;
    tipo: string;
  end;
  t_registro_reclamo2=record
    codigo: int16;
    anio: t_anio;
    tipo: string;
  end;
  t_lista_reclamos=^t_nodo_reclamos;
  t_nodo_reclamos=record
    ele: t_registro_reclamo2;
    sig: t_lista_reclamos;
  end;
  t_registro_dni=record
    dni: int8;
    reclamos: t_lista_reclamos;
    cantidad: int16;
  end;
  t_abb_dnis=^t_nodo_abb_dnis;
  t_nodo_abb_dnis=record
    ele: t_registro_dni;
    hi: t_abb_dnis;
    hd: t_abb_dnis;
  end;
  t_lista_codigos=^t_nodo_codigos2;
  t_nodo_codigos2=record
    ele: int16;
    sig: t_lista_codigos;
  end;
```

```
function random_string(length: int8): string;
var
  i: int8;
  string_aux: string;
begin
  string_aux:='';
  for i:= 1 to length do
    string_aux:=string_aux+chr(ord('A')+random(26));
    random_string:=string_aux;
  end;
end;

procedure leer_reclamo(var registro_reclamo1: t_registro_reclamo1);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_reclamo1.codigo:=codigo_salida
  else
    registro_reclamo1.codigo:=1+random(high(int16));
  if (registro_reclamo1.codigo<>codigo_salida) then
    begin
      registro_reclamo1.dni:=1+random(high(int8));
      registro_reclamo1.anio:=anio_ini+random(anio_fin-anio_ini+1);
      registro_reclamo1.tipo:=random_string(5+random(6));
    end;
  end;
end;

procedure cargar_registro_reclamo2(var registro_reclamo2: t_registro_reclamo2;
registro_reclamo1: t_registro_reclamo1);
begin
  registro_reclamo2.codigo:=registro_reclamo1.codigo;
  registro_reclamo2.anio:=registro_reclamo1.anio;
  registro_reclamo2.tipo:=registro_reclamo1.tipo;
end;

procedure agregar_adelante_lista_reclamos(var lista_reclamos: t_lista_reclamos;
registro_reclamo1: t_registro_reclamo1);
var
  nuevo: t_lista_reclamos;
begin
  new(nuevo);
  cargar_registro_reclamo2(nuevo^.ele,registro_reclamo1);
  nuevo^.sig:=lista_reclamos;
  lista_reclamos:=nuevo;
end;

procedure cargar_registro_dni(var registro_dni: t_registro_dni; registro_reclamo1:
t_registro_reclamo1);
begin
  registro_dni.dni:=registro_reclamo1.dni;
  registro_dni.reclamos:=nil;
  agregar_adelante_lista_reclamos(registro_dni.reclamos,registro_reclamo1);
  registro_dni.cantidad:=1;
end;

procedure agregar_abb_dnis(var abb_dnis: t_abb_dnis; registro_reclamo1: t_registro_reclamo1);
begin
  if (abb_dnis=nil) then
    begin
      new(abb_dnis);
      cargar_registro_dni(abb_dnis^.ele,registro_reclamo1);
      abb_dnis^.hi:=nil;
      abb_dnis^.hd:=nil;
    end
  else
    if (registro_reclamo1.dni=abb_dnis^.ele.dni) then
      begin
        agregar_adelante_lista_reclamos(abb_dnis^.ele.reclamos,registro_reclamo1);
        abb_dnis^.ele.cantidad:=abb_dnis^.ele.cantidad+1;
      end
    end
  end;
end
```

```

    else if (registro_reclamo1.dni<abb_dnis^.ele.dni) then
        agregar_abb_dnis(abb_dnis^.hi,registro_reclamo1)
    else
        agregar_abb_dnis(abb_dnis^.hd,registro_reclamo1);
end;
procedure cargar_abb_dnis(var abb_dnis: t_abb_dnis);
var
    registro_reclamo1: t_registro_reclamo1;
begin
    leer_reclamo(registro_reclamo1);
    while (registro_reclamo1.codigo<>codigo_salida) do
        begin
            agregar_abb_dnis(abb_dnis,registro_reclamo1);
            leer_reclamo(registro_reclamo1);
        end;
    end;
end;
procedure imprimir_registro_reclamo2(registro_reclamo2: t_registro_reclamo2; dni: int8;
reclamo: int16);
begin
    textcolor(green); write('El código de reclamo del reclamo '); textcolor(yellow);
write(reclamo); textcolor(green); write(' del DNI '); textcolor(yellow); write(dni);
textcolor(green); write(' es '); textcolor(red); writeln(registro_reclamo2.codigo);
    textcolor(green); write('El año del reclamo '); textcolor(yellow); write(reclamo);
textcolor(green); write(' del DNI '); textcolor(yellow); write(dni); textcolor(green); write('
es '); textcolor(red); writeln(registro_reclamo2.anio);
    textcolor(green); write('El tipo de reclamo del reclamo '); textcolor(yellow);
write(reclamo); textcolor(green); write(' del DNI '); textcolor(yellow); write(dni);
textcolor(green); write(' es '); textcolor(red); writeln(registro_reclamo2.tipo);
end;
procedure imprimir_lista_reclamos(lista_reclamos: t_lista_reclamos; dni: int8);
var
    i: int16;
begin
    i:=0;
    while (lista_reclamos<>nil) do
        begin
            i:=i+1;
            imprimir_registro_reclamo2(lista_reclamos^.ele,dni,i);
            lista_reclamos:=lista_reclamos^.sig;
        end;
    end;
end;
procedure imprimir_registro_dni(registro_dni: t_registro_dni);
begin
    textcolor(green); write('El DNI de la persona es '); textcolor(red);
writeln(registro_dni.dni);
    textcolor(green); write('La cantidad total de reclamos que realizó la persona es ');
textcolor(red); writeln(registro_dni.cantidad);
    imprimir_lista_reclamos(registro_dni.reclamos,registro_dni.dni);
    writeln();
end;
procedure imprimir_abb_dnis(abb_dnis: t_abb_dnis);
begin
    if (abb_dnis<>nil) then
        begin
            imprimir_abb_dnis(abb_dnis^.hi);
            imprimir_registro_dni(abb_dnis^.ele);
            imprimir_abb_dnis(abb_dnis^.hd);
        end;
    end;
end;
function contar_reclamos1(abb_dnis: t_abb_dnis; dni: int8): int16;
begin
    if (abb_dnis=nil) then
        contar_reclamos1:=0
    else
        if (dni=abb_dnis^.ele.dni) then
            contar_reclamos1:=abb_dnis^.ele.cantidad

```

```
    else if (dni<abb_dnis^.ele.dni) then
        contar_reclamos1:=contar_reclamos1(abb_dnis^.hi,dni)
    else
        contar_reclamos1:=contar_reclamos1(abb_dnis^.hd,dni);
end;
procedure verificar_dnis(var dni1, dni2: int8);
var
    aux: int8;
begin
    if (dni1>dni2) then
        begin
            aux:=dni1;
            dni1:=dni2;
            dni2:=aux;
        end;
    end;
end;
function contar_reclamos2(abb_dnis: t_abb_dnis; dni1, dni2: int8): int16;
begin
    if (abb_dnis=nil) then
        contar_reclamos2:=0
    else
        if (dni1>=abb_dnis^.ele.dni) then
            contar_reclamos2:=contar_reclamos2(abb_dnis^.hd,dni1,dni2)
        else if (dni2<=abb_dnis^.ele.dni) then
            contar_reclamos2:=contar_reclamos2(abb_dnis^.hi,dni1,dni2)
        else
            contar_reclamos2:=contar_reclamos2(abb_dnis^.hi,dni1,dni2)+contar_reclamos2(abb_dnis^.hd
,dni1,dni2)+1;
    end;
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: t_anio);
begin
    while (lista_reclamos<>nil) do
        begin
            if (anio=lista_reclamos^.ele.anio) then
                agregar_adelante_lista_codigos(lista_codigos,lista_reclamos^.ele.codigo);
            lista_reclamos:=lista_reclamos^.sig;
        end;
    end;
end;
procedure cargar_lista_codigos(var lista_codigos: t_lista_codigos; abb_dnis: t_abb_dnis; anio:
t_anio);
begin
    if (abb_dnis<>nil) then
        begin
            cargar_lista_codigos(lista_codigos,abb_dnis^.hd,anio);
            recorrer_lista_reclamos(lista_codigos,abb_dnis^.ele.reclamos,anio);
            cargar_lista_codigos(lista_codigos,abb_dnis^.hi,anio);
        end;
    end;
end;
procedure imprimir_lista_codigos(lista_codigos: t_lista_codigos; anio: t_anio);
var
    i: int16;
begin
    i:=0;
    while (lista_codigos<>nil) do
        begin
            i:=i+1;
```

```
    textcolor(green); write('Código de reclamo '); textcolor(yellow); write(i);
textcolor(green); write(' del año '); textcolor(yellow); write(anio); textcolor(green);
write(': '); textcolor(red); writeln(lista_codigos^.ele);
    lista_codigos:=lista_codigos^.sig;
end;
end;
var
    lista_codigos: t_lista_codigos;
    abb_dnis: t_abb_dnis;
    anio: t_anio;
    dni, dni1, dni2: int8;
begin
    randomize;
    abb_dnis:=nil;
    lista_codigos:=nil;
    writeln(); textcolor(red); writeln('INCISO (a):'); writeln();
    cargar_abb_dnis(abb_dnis);
    if (abb_dnis<>nil) then
    begin
        imprimir_abb_dnis(abb_dnis);
        writeln(); textcolor(red); writeln('INCISO (b):'); writeln();
        dni:=1+random(high(int8));
        textcolor(green); write('La cantidad de reclamos del DNI '); textcolor(yellow);
write(dni); textcolor(green); write(' es '); textcolor(red);
writeln(contar_reclamos1(abb_dnis,dni));
        writeln(); textcolor(red); writeln('INCISO (c):'); writeln();
        dni1:=1+random(high(int8)); dni2:=1+random(high(int8));
        verificar_dnis(dni1,dni2);
        textcolor(green); write('La cantidad de reclamos en el abb cuyo DNI se encuentra entre ');
textcolor(yellow); write(dni1); textcolor(green); write(' y '); textcolor(yellow);
write(dni2); textcolor(green); write(' es '); textcolor(red);
writeln(contar_reclamos2(abb_dnis,dni1,dni2));
        writeln(); textcolor(red); writeln('INCISO (d):'); writeln();
        anio:=anio_ini+random(anio_fin-anio_ini+1);
        cargar_lista_codigos(lista_codigos,abb_dnis,anio);
        if (lista_codigos<>nil) then
            imprimir_lista_codigos(lista_codigos,anio);
        end;
    end;
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
  anio_ini=2000; anio_fin=2023;
  codigo_salida=-1;
type
  t_anio=anio_ini..anio_fin;
  t_registro_reclamo1=record
    codigo: int16;
    dni: int32;
    anio: t_anio;
    tipo: string;
  end;
  t_registro_reclamo2=record
    codigo: int16;
    anio: t_anio;
    tipo: string;
  end;
  t_lista_reclamos=^t_nodo_reclamos;
  t_nodo_reclamos=record
    ele: t_registro_reclamo2;
    sig: t_lista_reclamos;
  end;
  t_registro_dni=record
    dni: int32;
    reclamos: t_lista_reclamos;
    cantidad: int16;
  end;
  t_abb_dnis=^t_nodo_abb_dnis;
  t_nodo_abb_dnis=record
    ele: t_registro_dni;
    hi: t_abb_dnis;
    hd: t_abb_dnis;
  end;
function random_string(length: int8): string;
var
  i: int8;
  string_aux: string;
begin
  string_aux:='';
  for i:= 1 to length do
    string_aux:=string_aux+chr(ord('A')+random(26));
  random_string:=string_aux;
end;
procedure leer_reclamo(var registro_reclamo1: t_registro_reclamo1; dni: int32);
var
  i: int8;
begin
  i:=random(100);
  if (i=0) then
    registro_reclamo1.codigo:=codigo_salida
  else
    registro_reclamo1.codigo:=1+random(high(int16));
  if (registro_reclamo1.codigo<>codigo_salida) then
    begin
      i:=random(2);
      if (i=0) then

```



```

        registro_reclamo1.dni:=dni
    else
        registro_reclamo1.dni:=10000000+random(40000001);
        registro_reclamo1.anio:=anio_ini+random(anio_fin-anio_ini+1);
        registro_reclamo1.tipo:=random_string(5+random(6));
    end;
end;
procedure cargar_registro_reclamo2(var registro_reclamo2: t_registro_reclamo2;
registro_reclamo1: t_registro_reclamo1);
begin
    registro_reclamo2.codigo:=registro_reclamo1.codigo;
    registro_reclamo2.anio:=registro_reclamo1.anio;
    registro_reclamo2.tipo:=registro_reclamo1.tipo;
end;
procedure agregar_adelante_lista_reclamos(var lista_reclamos: t_lista_reclamos;
registro_reclamo1: t_registro_reclamo1);
var
    nuevo: t_lista_reclamos;
begin
    new(nuevo);
    cargar_registro_reclamo2(nuevo^.ele,registro_reclamo1);
    nuevo^.sig:=lista_reclamos;
    lista_reclamos:=nuevo;
end;
procedure agregar_abb_dnis(var abb_dnis: t_abb_dnis; registro_dni: t_registro_dni);
begin
    if (abb_dnis=nil) then
        begin
            new(abb_dnis);
            abb_dnis^.ele:=registro_dni;
            abb_dnis^.hi:=nil;
            abb_dnis^.hd:=nil;
        end
    else
        if (registro_dni.dni<abb_dnis^.ele.dni) then
            agregar_abb_dnis(abb_dnis^.hi,registro_dni)
        else if (registro_dni.dni>abb_dnis^.ele.dni) then
            agregar_abb_dnis(abb_dnis^.hd,registro_dni);
        end;
    end;
end;
procedure cargar_abb_dnis(var abb_dnis: t_abb_dnis);
var
    registro_reclamo1: t_registro_reclamo1;
    registro_dni: t_registro_dni;
begin
    leer_reclamo(registro_reclamo1,10000000+random(40000001));
    while (registro_reclamo1.codigo<>codigo_salida) do
        begin
            registro_dni.dni:=registro_reclamo1.dni;
            registro_dni.reclamos:=nil;
            registro_dni.cantidad:=0;
            while ((registro_reclamo1.codigo<>codigo_salida) and
(registro_reclamo1.dni=registro_dni.dni)) do
                begin
                    agregar_adelante_lista_reclamos(registro_dni.reclamos,registro_reclamo1);
                    registro_dni.cantidad:=registro_dni.cantidad+1;
                    leer_reclamo(registro_reclamo1,registro_dni.dni);
                end;
            agregar_abb_dnis(abb_dnis,registro_dni);
        end;
    end;
end;
procedure imprimir_registro_reclamo2(registro_reclamo2: t_registro_reclamo2; dni: int32;
reclamo: int16);
begin
    textcolor(green); write('El código de reclamo del reclamo '); textcolor(yellow);
write(reclamo); textcolor(green); write(' del DNI '); textcolor(yellow); write(dni);
textcolor(green); write(' es '); textcolor(red); writeln(registro_reclamo2.codigo);

```

```
    textcolor(green); write('El año del reclamo '); textcolor(yellow); write(reclamo);
textcolor(green); write(' del DNI '); textcolor(yellow); write(dni); textcolor(green); write('
es '); textcolor(red); writeln(registro_reclamo2.anio);
    textcolor(green); write('El tipo de reclamo del reclamo '); textcolor(yellow);
write(reclamo); textcolor(green); write(' del DNI '); textcolor(yellow); write(dni);
textcolor(green); write(' es '); textcolor(red); writeln(registro_reclamo2.tipo);
end;
procedure imprimir_lista_reclamos(lista_reclamos: t_lista_reclamos; dni: int32);
var
    i: int16;
begin
    i:=0;
    while (lista_reclamos<>nil) do
        begin
            i:=i+1;
            imprimir_registro_reclamo2(lista_reclamos^.ele,dni,i);
            lista_reclamos:=lista_reclamos^.sig;
        end;
    end;
procedure imprimir_registro_dni(registro_dni: t_registro_dni);
begin
    textcolor(green); write('El DNI de la persona es '); textcolor(red);
writeln(registro_dni.dni);
    textcolor(green); write('La cantidad total de reclamos que realizó la persona es ');
textcolor(red); writeln(registro_dni.cantidad);
    imprimir_lista_reclamos(registro_dni.reclamos,registro_dni.dni);
    writeln();
end;
procedure imprimir_abb_dnis(abb_dnis: t_abb_dnis);
begin
    if (abb_dnis<>nil) then
        begin
            imprimir_abb_dnis(abb_dnis^.hi);
            imprimir_registro_dni(abb_dnis^.ele);
            imprimir_abb_dnis(abb_dnis^.hd);
        end;
    end;
var
    abb_dnis: t_abb_dnis;
begin
    randomize;
    abb_dnis:=nil;
    cargar_abb_dnis(abb_dnis);
    imprimir_abb_dnis(abb_dnis);
end.
```

Trabajo Práctico N° 6: **Módulo Objetos (Introducción a Java. Matrices).**

Ejercicio 1.

Analizar el programa Ej01Tabla2.java, que carga un vector que representa la tabla del 2. Luego, escribir las instrucciones necesarias para generar enteros aleatorios hasta obtener el número 11. Para cada número, mostrar el resultado de multiplicarlo por 2 (accediendo al vector).

Ejercicio 2.

Escribir un programa que lea las alturas de los 15 jugadores de un equipo de básquet y las almacene en un vector. Luego, informar:

- *La altura promedio.*
- *La cantidad de jugadores con altura por encima del promedio.*

NOTA: Se dispone de un esqueleto para este programa en Ej02Jugadores.java.

Ejercicio 3.

Escribir un programa que defina una matriz de enteros de tamaño 5x5. Inicializar la matriz con números aleatorios entre 0 y 30. Luego, realizar las siguientes operaciones:

- *Mostrar el contenido de la matriz en consola.*
- *Calcular e informar la suma de los elementos de la fila 1.*
- *Generar un vector de 5 posiciones donde cada posición j contiene la suma de los elementos de la columna j de la matriz. Luego, imprimir el vector.*
- *Leer un valor entero e indicar si se encuentra o no en la matriz. En caso de encontrarse, indicar su ubicación (fila y columna), en caso contrario imprimir "No se encontró el elemento".*

NOTA: Se dispone de un esqueleto para este programa en Ej03Matrices.java.

Ejercicio 4.

Un edificio de oficinas está conformado por 8 pisos (1..8) y 4 oficinas por piso (1..4). Realizar un programa que permita informar la cantidad de personas que concurrieron a cada oficina de cada piso. Para esto, simular la llegada de personas al edificio de la siguiente manera: a cada persona, se le pide el nro. de piso y nro. de oficina a la cual quiere concurrir. La llegada de personas finaliza al indicar un nro. de piso 9. Al finalizar la llegada de personas, informar lo pedido.

Ejercicio 5.

El dueño de un restaurante entrevista a cinco clientes y les pide que califiquen (con puntaje de 1 a 10) los siguientes aspectos: (0) Atención al cliente, (1) Calidad de la comida, (2) Precio, (3) Ambiente. Escribir un programa que lea desde teclado las calificaciones de los cinco clientes para cada uno de los aspectos y almacene la información en una estructura. Luego, imprima la calificación promedio obtenida por cada aspecto.

Trabajo Práctico N° 7: **Módulo Objetos (Introducción a POO).**

Ejercicio 1.

Se dispone de la clase Persona (en la carpeta tema2). Un objeto persona puede crearse sin valores iniciales o enviando, en el mensaje de creación, el nombre, DNI y edad (en ese orden). Un objeto persona responde a los siguientes mensajes:

- *getNombre() retorna el nombre (String) de la persona;*
- *getDNI() retorna el dni (int) de la persona;*
- *getEdad() retorna la edad (int) de la persona;*
- *setNombre(X) modifica el nombre de la persona al “String” pasado por parámetro (X);*
- *setDNI(X) modifica el DNI de la persona al “int” pasado por parámetro (X);*
- *setEdad(X) modifica la edad de la persona al “int” pasado por parámetro (X);*
- *toString() retorna un String que representa al objeto. Ejemplo: “Mi nombre es Mauro, mi DNI es 11203737 y tengo 70 años”.*

Realizar un programa que cree un objeto persona con datos leídos desde teclado. Luego, mostrar en consola la representación de ese objeto en formato String.

Ejercicio 2.

Utilizando la clase Persona, realizar un programa que almacene en un vector, a lo sumo, 15 personas. La información (nombre, DNI, edad) se debe generar aleatoriamente hasta obtener edad 0. Luego de almacenar la información:

- *Informar la cantidad de personas mayores de 65 años.*
- *Mostrar la representación de la persona con menor DNI.*

Ejercicio 3.

Se realizará un casting para un programa de TV. El casting durará, a lo sumo, 5 días y, en cada día, se entrevistarán a 8 personas en distinto turno.

(a) Simular el proceso de inscripción de personas al casting. A cada persona, se le pide nombre, DNI y edad y se la debe asignar en un día y turno de la siguiente manera: las personas, primero, completan el primer día en turnos sucesivos, luego el segundo día y así siguiendo. La inscripción finaliza al llegar una persona con nombre “ZZZ” o al cubrirse los 40 cupos de casting.

(b) Una vez finalizada la inscripción, informar, para cada día y turno asignado, el nombre de la persona a entrevistar. NOTA: Utilizar la clase Persona. Pensar en la estructura de datos a utilizar. Para comparar Strings, usar el método equals.

Ejercicio 4.

Sobre un nuevo programa, modificar el ejercicio anterior para considerar que:

(a) *Durante el proceso de inscripción, se pida a cada persona sus datos (nombre, DNI, edad) y el día en que se quiere presentar al casting. La persona debe ser inscripta en ese día en el siguiente turno disponible. En caso de no existir un turno en ese día, informar la situación. La inscripción finaliza al llegar una persona con nombre “ZZZ” o al cubrirse los 40 cupos de casting.*

(b) *Una vez finalizada la inscripción, informar para cada día: la cantidad de inscriptos al casting ese día y el nombre de la persona a entrevistar en cada turno asignado.*

Ejercicio 5.

Se dispone de la clase Partido (en la carpeta tema2). Un objeto partido representa un encuentro entre dos equipos (local y visitante). Un objeto partido puede crearse sin valores iniciales o enviando, en el mensaje de creación, el nombre del equipo local, el nombre del visitante, la cantidad de goles del local y del visitante (en ese orden). Un objeto partido sabe responder a los siguientes mensajes:

- *getLocal()* retorna el nombre (String) del equipo local;
- *getVisitante()* retorna el nombre (String) del equipo visitante;
- *getGolesLocal()* retorna la cantidad de goles (int) del equipo local;
- *getGolesVisitante()* retorna la cantidad de goles (int) del equipo visitante;
- *setLocal(X)* modifica el nombre del equipo local al "String" X;
- *setVisitante(X)* modifica el nombre del equipo visitante al "String" X;
- *setGolesLocal(X)* modifica la cantidad de goles del equipo local al "int" X;
- *setGolesVisitante(X)* modifica la cantidad de goles del equipo visitante al "int" X;
- *hayGanador()* retorna un boolean que indica si hubo (true) o no hubo (false) ganador;
- *getGanador()* retorna el nombre (String) del ganador del partido (si no hubo retorna un String vacío);
- *hayEmpate()* retorna un boolean que indica si hubo (true) o no hubo (false) empate;

Implementar un programa que cargue un vector con, a lo sumo, 20 partidos disputados en el campeonato. La información de cada partido se lee desde teclado hasta ingresar uno con nombre de visitante "ZZZ" o alcanzar los 20 partidos. Luego de la carga:

- *Para cada partido, armar e informar una representación String del estilo: {EQUIPO-LOCAL golesLocal VS EQUIPO-VISITANTE golesVisitante}.*
- *Calcular e informar la cantidad de partidos que ganó River.*
- *Calcular e informar el total de goles que realizó Boca jugando de local.*

Trabajo Práctico N° 8: **Módulo Objetos (Desarrollo de Clases).**

Ejercicio 1.

(a) Definir una clase para representar triángulos. Un triángulo se caracteriza por el tamaño de sus 3 lados (*double*), el color de relleno (*String*) y el color de línea (*String*). Proveer un constructor que reciba todos los datos necesarios para iniciar el objeto. Proveer métodos para:

- Devolver/modificar el valor de cada uno de sus atributos (métodos *get* y *set*).
- Calcular el perímetro y devolverlo (método *calcularPerimetro*).
- Calcular el área y devolverla (método *calcularArea*).

NOTA: Calcular el área con la fórmula $\text{Área} = \sqrt{s(s-a)(s-b)(s-c)}$, donde *a*, *b* y *c* son lados y $s = \frac{a+b+c}{2}$. La función raíz cuadrada es *Math.sqrt(#)*.

(b) Realizar un programa que instancie un triángulo, le cargue información leída desde teclado e informe en consola el perímetro y el área.

Ejercicio 2.

(a) *Modificar la clase Libro.java (carpeta tema3) para, ahora, considerar que el primer autor es un objeto instancia de la clase Autor. Implementar la clase Autor, sabiendo que se caracterizan por nombre, biografía y origen y que deben permitir devolver/modificar el valor de sus atributos y devolver una representación String formada por nombre, biografía y origen. Luego, realizar las modificaciones necesarias en la clase Libro.*

(b) *Modificar el programa Demo01Constructores (carpeta tema3) para instanciar los libros con su autor, considerando las modificaciones realizadas. Luego, a partir de uno de los libros instanciados, obtener e imprimir la representación del autor de ese libro.*

Ejercicio 3.

(a) Definir una clase para representar estantes. Un estante almacena, a lo sumo, 20 libros. Implementar un constructor que permita iniciar el estante sin libros. Proveer métodos para:

- Devolver la cantidad de libros que hay almacenados.
- Devolver si el estante está lleno.
- Agregar un libro al estante.
- Devolver el libro con un título particular que se recibe.

(b) Realizar un programa que instancie un estante. Cargar varios libros. A partir del estante, buscar e informar el autor del libro “Mujercitas”.

(c) ¿Qué se modificaría en la clase definida para, ahora, permitir estantes que almacenen como máximo N libros? ¿Cómo se instanciaría el estante?

Ejercicio 4.

(a) *Un hotel posee N habitaciones. De cada habitación, se conoce costo por noche, si está ocupada y, en caso de estarlo, guarda el cliente que la reservó (nombre, DNI y edad).*

(i) *Generar las clases necesarias. Para cada una, proveer métodos getters/setters adecuados.*

(ii) *Implementar los constructores necesarios para iniciar: los clientes a partir de nombre, DNI, edad; el hotel para N habitaciones, cada una desocupada y con costo aleatorio entre 2.000 y 8.000.*

(iii) *Implementar, en las clases que corresponda, todos los métodos necesarios para:*

- *Ingresar un cliente C en la habitación número X . Asumir que X es válido (es decir, está en el rango $1..N$) y que la habitación está libre.*
- *Aumentar el precio de todas las habitaciones en un monto recibido.*
- *Obtener la representación String del hotel, siguiendo el formato:
{Habitación 1: costo, libre u ocupada, información del cliente si está ocupada},
...
{Habitación N : costo, libre u ocupada, información del cliente si está ocupada}.*

(b) *Realizar un programa que instancie un hotel, ingrese clientes en distintas habitaciones, muestre el hotel, aumente el precio de las habitaciones y vuelva a mostrar el hotel.*

NOTA: Reusar la clase Persona. Para cada método solicitado, pensar a qué clase debe delegar la responsabilidad de la operación.

Ejercicio 5.

(a) Definir una clase para representar círculos. Los círculos se caracterizan por su radio (*double*), el color de relleno (*String*) y el color de línea (*String*). Proveer un constructor que reciba todos los datos necesarios para iniciar el objeto. Proveer métodos para:

- Devolver/modificar el valor de cada uno de sus atributos (métodos *get* y *set*).
- Calcular el perímetro y devolverlo (método *calcularPerimetro*).
- Calcular el área y devolverla (método *calcularArea*).

(b) Realizar un programa que instancie un círculo, le cargue información leída de teclado e informe en consola el perímetro y el área.

NOTA: La constante π es `Math.PI`.

Trabajo Práctico N° 9: **Módulo Objetos (Herencia).**

Ejercicio 1.

(a) *Incluir la clase Triángulo a la jerarquía de figuras vista (carpeta tema4). Triángulo debe heredar de Figura todo lo que es común y definir su constructor y sus atributos y métodos propios. Además, se debe redefinir el método toString.*

(b) *De igual manera, incluir la clase Círculo a la jerarquía de figuras.*

(c) *Añadir, a la representación String, el valor del perímetro. Pensar: ¿qué método toString se debe modificar: el de cada subclase o el de Figura?*

(d) *Añadir el método despintar que establece los colores de la figura a línea “negra” y relleno “blanco”. Pensar: ¿dónde se debe definir el método: en cada subclase o en Figura?*

(e) *Realizar un programa que instancie un triángulo y un círculo. Mostrar en consola la representación String de cada uno. Probar el funcionamiento del método despintar.*

Ejercicio 2.

Se quiere representar a los empleados de un club: jugadores y entrenadores.

- *Cualquier empleado se caracteriza por su nombre, sueldo básico y antigüedad.*
- *Los jugadores son empleados que se caracterizan por el número de partidos jugados y el número de goles anotados.*
- *Los entrenadores son empleados que se caracterizan por la cantidad de campeonatos ganados.*

(a) *Implementar la jerarquía de clases declarando atributos, métodos para obtener/modificar su valor y constructores que reciban los datos necesarios.*

(b) *Cualquier empleado debe responder al mensaje calcularEfectividad. La efectividad del entrenador es el promedio de campeonatos ganados por año de antigüedad, mientras que la del jugador es el promedio de goles por partido.*

(c) *Cualquier empleado debe responder al mensaje calcularSueldoACobrar. El sueldo a cobrar es el sueldo básico más un 10% del básico por cada año de antigüedad y, además:*

- *Para los jugadores: si el promedio de goles por partido es superior a 0,5, se adiciona un plus de otro sueldo básico.*
- *Para los entrenadores: se adiciona un plus por campeonatos ganados (\$5.000 si ha ganado entre 1 y 4 campeonatos; \$30.000 si ha ganado entre 5 y 10 campeonatos; \$50.000 si ha ganado más de 10 campeonatos).*

(d) *Cualquier empleado debe responder al mensaje toString, que devuelve un String que lo representa, compuesto por nombre, sueldo a cobrar y efectividad.*

(f) *Realizar un programa que instancie un jugador y un entrenador. Informar la representación String de cada uno.*

NOTA: Para cada método a implementar, pensar en qué clase/s se debe definir el método.

Ejercicio 3.

(a) Implementar las clases para el siguiente problema. Una garita de seguridad quiere identificar los distintos tipos de personas que entran a un barrio cerrado. Al barrio, pueden entrar: personas, que se caracterizan por nombre, DNI y edad; y trabajadores, estos son personas que se caracterizan, además, por la tarea realizada en el predio. Implementar constructores, getters y setters para las clases. Además, tanto las personas como los trabajadores, deben responder al mensaje `toString` siguiendo el formato:

- *Personas:* “Mi nombre es Mauro, mi DNI es 11.203.737 y tengo 70 años.”
- *Trabajadores:* “Mi nombre es Mauro, mi DNI es 11.203.737 y tengo 70 años. Soy jardinero.”

(b) Realizar un programa que instancie una persona y un trabajador y mostrar la representación de cada uno en consola.

NOTA: Reutilizar la clase *Persona* (carpeta tema2).

Ejercicio 4.

Un objeto visor de figuras se encarga de mostrar en consola cualquier figura que reciba y también mantiene cuántas figuras mostró. Analizar y ejecutar el siguiente programa y responder: ¿Qué imprime? ¿Por qué?

Ejercicio 5.

(a) *Modificar la clase VisorFiguras: ahora, debe permitir guardar las figuras a mostrar (a lo sumo, 5) y también mostrar todas las figuras guardadas. Usar la siguiente estructura.*

(b) *Realizar un programa que instancie el visor, guarde dos cuadrados y un rectángulo en el visor y, por último, haga que el visor muestre sus figuras almacenadas.*

Ejercicio 6.

El Servicio Meteorológico Nacional necesita un sistema que permita registrar, para una determinada estación meteorológica, la temperatura promedio mensual de N años consecutivos a partir de un año A dado. Además, necesita dos versiones del sistema: una que permita reportar el promedio histórico por años y otra que permita reportar el promedio histórico por meses. Esto se detalla más adelante. De la estación, interesa conocer: nombre y latitud y longitud donde se encuentra. Implementar las clases, constructores y métodos que se considere necesarios para:

(a) *Crear el sistema de registro/reporte, que funcionará en una determinada estación, para N años consecutivos a partir de un año A . Iniciar cada temperatura en un valor muy alto.*

(b) *Registrar la temperatura de un mes y año recibidos por parámetro. Nota: El mes está en rango 1..12 y el año está en rango $A..A+N-1$.*

(c) *Obtener la temperatura de un mes y año recibidos por parámetro. Nota: El mes está en rango 1..12 y el año está en rango $A..A+N-1$. En caso de no haberse registrado temperatura para ese mes/año, se retorna el valor muy alto.*

(d) *Devolver un String que concatena el mes y año en que se registró la mayor temperatura. Nota: Suponer que ya están registradas las temperaturas de todos los meses y años.*

(e) *Devolver un String con el nombre de la estación, su latitud y longitud y los promedios mensuales o anuales según corresponda:*

- *La versión del sistema que reporta por años deberá calcular el promedio para cada año (el promedio del año X se calcula con los datos mensuales de ese año).*
- *La versión del sistema que reporta por meses deberá calcular el promedio para cada mes (el promedio del mes M se calcula con los datos de todos los años en ese mes).*

NOTA: Suponer que ya están registradas las temperaturas de todos los meses y años. Utilizar el carácter `\n` para concatenar un salto de línea.

(f) *Realizar un programa principal que cree un Sistema con reporte anual para 3 años consecutivos a partir del 2021, para la estación La Plata (latitud -34.921 y longitud -57.955). Cargar todas las temperaturas (para todos los meses y años). Informar los promedios anuales y el mes y año en que se registró la mayor temperatura. Luego, crear un Sistema con informe mensual para 4 años a partir de 2020, para la estación Mar del Plata (latitud -38.002 y longitud -57.556). Cargar todas las temperaturas (para todos los meses y años). Informar los promedios mensuales y el mes y año en que se registró la mayor temperatura.*

NOTA: Prestar atención de no violar el encapsulamiento al resolver el ejercicio.

Trabajo Práctico N° 10: **Módulo Objetos (Repaso).**

Ejercicio 1.

La UNLP desea administrar sus proyectos, investigadores y subsidios. Un proyecto tiene: nombre, código, nombre completo del director y los investigadores que participan en el proyecto (50 como máximo). De cada investigador, se tiene: nombre completo, categoría (1 a 5) y especialidad. Además, cualquier investigador puede pedir hasta un máximo de 5 subsidios. De cada subsidio, se conoce: el monto pedido, el motivo y si fue otorgado o no.

(a) *Implementar el modelo de clases teniendo en cuenta:*

- *Un proyecto sólo debería poder construirse con el nombre, código, nombre del director.*
- *Un investigador sólo debería poder construirse con nombre, categoría y especialidad.*
- *Un subsidio sólo debería poder construirse con el monto pedido y el motivo. Un subsidio siempre se crea en estado no-otorgado.*

(b) *Implementar los métodos necesarios (en las clases donde corresponda) que permitan:*

- *void agregarInvestigador(Investigador unInvestigador);
// agregar un investigador al proyecto.*
- *void agregarSubsidio(Subsidio unSubsidio);
// agregar un subsidio al investigador.*
- *double dineroTotalOtorgado();
// devolver el monto total otorgado en subsidios del proyecto (tener en cuenta todos los subsidios otorgados de todos los investigadores).*
- *void otorgarTodos(String nombre_completo);
// otorgar todos los subsidios no-otorgados del investigador llamado nombre_completo.*
- *String toString();
// devolver un string con: nombre del proyecto, código, nombre del director, el total de dinero otorgado del proyecto y la siguiente información de cada investigador: nombre, categoría, especialidad y el total de dinero de sus subsidios otorgados.*

(c) *Escribir un programa que instancie un proyecto con tres investigadores. Agregar dos subsidios a cada investigador y otorgar los subsidios de uno de ellos. Luego, imprimir todos los datos del proyecto en pantalla.*

Ejercicio 2.

Se quiere un sistema para gestionar estacionamientos. Un estacionamiento conoce su nombre, dirección, hora de apertura, hora de cierre y almacena, para cada número de piso (1..N) y número de plaza (1..M), el auto que ocupa dicho lugar. De los autos, se conoce nombre del dueño y patente.

(a) *Generar las clases, incluyendo getters y setters adecuados.*

(b) *Implementar constructores. En particular, para el estacionamiento:*

- *Un constructor debe recibir nombre y dirección e iniciar el estacionamiento con hora de apertura "8:00", hora de cierre "21:00" y para 5 pisos y 10 plazas por piso. El estacionamiento inicialmente no tiene autos.*
- *Otro constructor debe recibir nombre, dirección, hora de apertura, hora de cierre, el número de pisos (N) y el número de plazas por piso (M) e iniciar el estacionamiento con los datos recibidos y sin autos.*

(c) *Implementar métodos para:*

- *Dado un auto A, un número de piso X y un número de plaza Y, registrar al auto en el estacionamiento en el lugar (X, Y). Suponer que (X, Y) son válidos (es decir, están en rango 1..N y 1..M, respectivamente) y que el lugar está desocupado.*
- *Dada una patente, obtener un String que contenga el número de piso y plaza donde está dicho auto en el estacionamiento. En caso de no encontrarse, retornar el mensaje "Auto Inexistente".*
- *Obtener un String con la representación del estacionamiento.
Ejemplo: "Piso 1 Plaza 1: libre; Piso 1 Plaza 2: representación del auto; ...; Piso 2 Plaza 1: libre; ..."*
- *Dado un número de plaza Y, obtener la cantidad de autos ubicados en dicha plaza (teniendo en cuenta todos los pisos).*

(d) *Realizar un programa que instancie un estacionamiento con 3 pisos y 3 plazas por piso. Registrar 6 autos en el estacionamiento en distintos lugares. Mostrar la representación String del estacionamiento en consola. Mostrar la cantidad de autos ubicados en la plaza 1. Leer una patente por teclado e informar si dicho auto se encuentra en el estacionamiento o no. En caso de encontrarse, la información a imprimir es el piso y plaza que ocupa.*

Ejercicio 3.

Un productor musical desea administrar los recitales que organiza, que pueden ser: eventos ocasionales y giras.

- *De todo recital, se conoce el nombre de la banda y la lista de temas que tocarán durante el recital.*
- *Un evento ocasional es un recital que, además, tiene el motivo (a beneficio, show de TV o show privado), el nombre del contratante del recital y el día del evento.*
- *Una gira es un recital que, además, tiene un nombre y las “fechas” donde se repetirá la actuación. De cada “fecha”, se conoce la ciudad y el día. Además, la gira guarda el número de la fecha en la que se tocará próximamente (actual).*

(a) *Generar las clases necesarias. Implementar métodos getters/setters adecuados.*

(b) *Implementar los constructores. El constructor de recitales recibe el nombre de la banda y la cantidad de temas que tendrá el recital. El constructor de eventos ocasionales, además, recibe el motivo, el nombre del contratante y día del evento. El constructor de giras, además, recibe el nombre de la gira y la cantidad de fechas que tendrá.*

(c) *Implementar los métodos listados a continuación:*

(i) *Cualquier recital debe saber responder a los mensajes:*

- **agregarTema** que recibe el nombre de un tema y lo agrega a la lista de temas.
- **actuar** que imprime (por consola), para cada tema, la leyenda “y ahora tocaremos...” seguido por el nombre del tema.

(ii) *La gira debe saber responder a los mensajes:*

- **agregarFecha** que recibe una “fecha” y la agrega adecuadamente.
- *La gira debe responder al mensaje **actuar** de manera distinta. Imprimir la leyenda “Buenas noches...” seguido del nombre de la ciudad de la fecha “actual”. Luego, debe imprimir el listado de temas como lo hace cualquier recital. Además, debe establecer la siguiente fecha de la gira como la nueva “actual”.*

(iii) *El evento ocasional debe saber responder al mensaje actuar de manera distinta:*

- *Si es un show de beneficencia, se imprime la leyenda “Recuerden colaborar con...” seguido del nombre del contratante.*
- *Si es un show de TV, se imprime “Saludos amigos televidentes”.*
- *Si es un show privado, se imprime “Un feliz cumpleaños para...” seguido del nombre del contratante.*

Independientemente del motivo del evento, luego, se imprime el listado de temas como lo hace cualquier recital.

(iv) *Todo recital debe saber responder al mensaje **calcularCosto** teniendo en cuenta lo siguiente. Si es un evento ocasional, devuelve 0 si es a beneficio, 50.000 si es un show de TV y 150.000 si es privado. Las giras deben devolver 30.000 por cada fecha de la misma.*

(d) *Realizar un programa que instancie un evento ocasional y una gira, cargando la información necesaria. Luego, para ambos, imprimir el costo e invocar al mensaje “actuar”.*

Ejercicio 4.

Una escuela de música arma coros para participar de ciertos eventos. Los coros poseen un nombre y están formados por un director y una serie de coristas. Del director, se conoce el nombre, DNI, edad y la antigüedad (un número entero). De los coristas, se conoce el nombre, DNI, edad y el tono fundamental (un número entero). Asimismo, hay dos tipos de coros: coro semicircular en el que los coristas se colocan en el escenario uno al lado del otro y coro por hileras donde los coristas se organizan en filas de igual dimensión.

(a) *Implementar las clases necesarias teniendo en cuenta que los coros deberían crearse con un director y sin ningún corista, pero sí sabiendo las dimensiones del coro.*

(b) *Implementar métodos (en las clases donde corresponda) que permitan:*

- *agregar un corista al coro.*
 - *En el coro semicircular, los coristas se deben ir agregando de izquierda a derecha.*
 - *En el coro por hileras, los coristas se deben ir agregando de izquierda a derecha, completando la hilera antes de pasar a la siguiente.*
- *determinar si un coro está lleno o no. Devuelve true si el coro tiene a todos sus coristas asignados o false en caso contrario.*
- *determinar si un coro (se supone que está lleno) está bien formado. Un coro está bien formado si:*
 - *En el caso del coro semicircular, de izquierda a derecha, los coristas están ordenados de mayor a menor en cuanto a tono fundamental.*
 - *En el caso del coro por hileras, todos los miembros de una misma hilera tienen el mismo tono fundamental y, además, todos los primeros miembros de cada hilera están ordenados de mayor a menor en cuanto a tono fundamental.*
- *devolver la representación de un coro formada por el nombre del coro, todos los datos del director y todos los datos de todos los coristas.*

(c) *Escribir un programa que instancie un coro de cada tipo. Leer o bien la cantidad de coristas (en el caso del coro semicircular) o la cantidad de hileras e integrantes por hilera (en el caso del coro por hileras). Luego, crear la cantidad de coristas necesarios, leyendo sus datos y almacenándolos en el coro. Finalmente, imprimir toda la información de los coros indicando si están bien formados o no.*

Trabajo Práctico N° 11:

Módulo Concurrente (Realizar programas en R-info en los que múltiples robots realizan tareas. Diseñar soluciones con robots del mismo tipo y con robots de diferentes tipos. Analizar situaciones de posibles colisiones).

Ejercicio 1.

(a) Realizar un programa para que un robot junte todas las flores de la avenida 1 y las deposite al final de dicha avenida. Al finalizar, debe informar la cantidad de flores depositadas y la cantidad de esquinas sin flores que encontró durante el recorrido.

programa TP11_E1a

procesos

proceso juntarFloresEsquina (ES flores: numero; ES esquinas: numero)

comenzar

si (HayFlorEnLaEsquina)

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

sino

esquinas:=esquinas+1

fin

proceso juntarFloresAvenida (ES flores: numero; ES esquinas: numero)

comenzar

repetir 99

juntarFloresEsquina(flores,esquinas)

mover

juntarFloresEsquina(flores,esquinas)

fin

proceso depositarFlores (E flores: numero)

comenzar

repetir flores

depositarFlor

fin

proceso recorrerAvenida

variables

flores, esquinas: numero

comenzar

flores:=0

esquinas:=0

juntarFloresAvenida(flores,esquinas)

depositarFlores(flores)

Informar ('Av_FloresDepositadas',PosAv,flores)

Informar ('Av_EsquinasVacías',PosAv,esquinas)

fin

areas

```

ciudad: AreaC(1,1,100,100)
robots
  robot recolector
  comenzar
    recorrerAvenida
  fin
variables
  robot1: recolector
comenzar
  AsignarArea(robot1,ciudad)
  Iniciar(robot1,1,1)
fin

```

(b) Modificar el programa anterior para que el mismo robot realice lo mismo en las avenidas 1, 3 y 5.

```

programa TP11_E1b
procesos
  proceso juntarFloresEsquina (ES flores: numero; ES esquinas: numero)
  comenzar
    si (HayFlorEnLaEsquina)
      mientras (HayFlorEnLaEsquina)
        tomarFlor
        flores:=flores+1
      sino
        esquinas:=esquinas+1
    fin
  proceso juntarFloresAvenida (ES flores: numero; ES esquinas: numero)
  comenzar
    repetir 99
      juntarFloresEsquina(flores,esquinas)
      mover
      juntarFloresEsquina(flores,esquinas)
    fin
  proceso depositarFlores (E flores: numero)
  comenzar
    repetir flores
      depositarFlor
    fin
  proceso recorrerAvenida
variables
  flores, esquinas: numero
comenzar
  flores:=0
  esquinas:=0
  juntarFloresAvenida(flores,esquinas)
  depositarFlores(flores)
  Informar ('Av_FloresDepositadas',PosAv,flores)

```

```

    Informar ('Av_EsquinasVacías',PosAv,esquinas)
fin
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot recolector
comenzar
    repetir 2
        recorrerAvenida
        Pos (PosAv+2,1)
    recorrerAvenida
fin
variables
    robot1: recolector
comenzar
    AsignarArea(robot1,ciudad)
    Iniciar(robot1,1,1)
fin

```

(c) Modificar el programa anterior para que el trabajo sea realizado por 3 robots: uno realiza la avenida 1, otro realiza la avenida 3 y otro la avenida 5. Cada robot debe iniciar en las esquinas (1,1), (3,1) y (5,1), respectivamente.

```

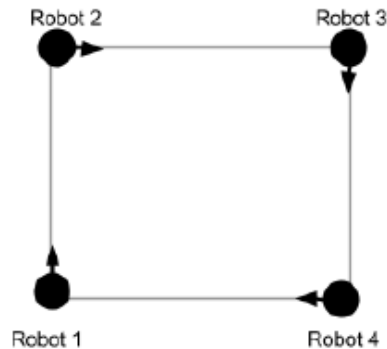
programa TP11_E1c
procesos
    proceso juntarFloresEsquina (ES flores: numero; ES esquinas: numero)
    comenzar
        si (HayFlorEnLaEsquina)
            mientras (HayFlorEnLaEsquina)
                tomarFlor
                flores:=flores+1
            sino
                esquinas:=esquinas+1
        fin
    proceso juntarFloresAvenida (ES flores: numero; ES esquinas: numero)
    comenzar
        repetir 99
            juntarFloresEsquina(flores,esquinas)
            mover
            juntarFloresEsquina(flores,esquinas)
        fin
    proceso depositarFlores (E flores: numero)
    comenzar
        repetir flores
            depositarFlor
        fin
    proceso recorrerAvenida
variables

```

```
flores, esquinas: numero
comenzar
  flores:=0
  esquinas:=0
  juntarFloresAvenida(flores,esquinas)
  depositarFlores(flores)
  Informar ('Av_FloresDepositadas',PosAv,flores)
  Informar ('Av_EsquinasVacías',PosAv,esquinas)
fin
areas
  area1: AreaP(1,1,1,100)
  area2: AreaP(3,1,3,100)
  area3: AreaP(5,1,5,100)
robots
  robot recolector
  comenzar
    recorrerAvenida
  fin
variables
  robot1, robot2, robot3: recolector
comenzar
  AsignarArea(robot1,area1)
  AsignarArea(robot2,area2)
  AsignarArea(robot3,area3)
  Iniciar(robot1,1,1)
  Iniciar(robot2,3,1)
  Iniciar(robot3,5,1)
fin
```


Ejercicio 2.

Realizar un programa en el que 4 robots limpien de papeles el perímetro de un cuadrado de lado 20 en sentido horario, como se muestra en la figura:



El vértice inferior izquierdo del cuadrado se ubica en la esquina (10,10). Al finalizar, cada robot debe informar la cantidad de papeles juntados en su lado. Al realizar este programa, analizar: ¿Cómo deben declararse la o las áreas? ¿Existe riesgo de colisión?

programa TP11_E2

procesos

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

proceso juntarPapelesArea (E lado: numero)

variables

papeles: numero

comenzar

papeles:=0

repetir 19

juntarPapelesEsquina(papeles)

mover

juntarPapelesEsquina(papeles)

Informar ('Lado_Papeles',lado,papeles)

fin

areas

area1: AreaP(10,10,10,29)

area2: AreaP(10,30,29,30)

area3: AreaP(30,11,30,30)

area4: AreaP(11,10,30,10)

robots

robot limpiador1

comenzar

juntarPapelesArea(1)

fin

```
robot limpiador2
comenzar
  derecha
  juntarPapelesArea(2)
fin
```

```
robot limpiador3
comenzar
  repetir 2
    derecha
  juntarPapelesArea(3)
fin
```

```
robot limpiador4
comenzar
  repetir 3
    derecha
  juntarPapelesArea(4)
fin
```

variables

```
robot1: limpiador1
robot2: limpiador2
robot3: limpiador3
robot4: limpiador4
```

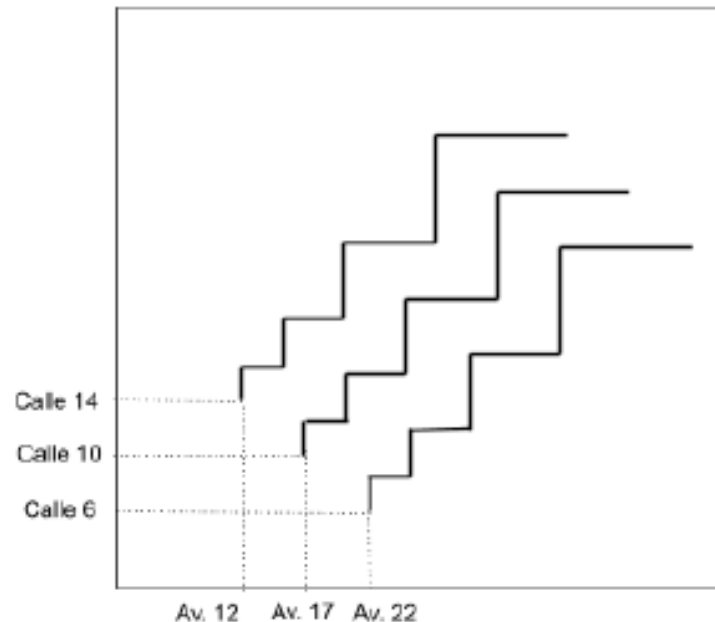
comenzar

```
AsignarArea(robot1,area1)
AsignarArea(robot2,area2)
AsignarArea(robot3,area3)
AsignarArea(robot4,area4)
Iniciar(robot1,10,10)
Iniciar(robot2,10,30)
Iniciar(robot3,30,30)
Iniciar(robot4,30,10)
```

fin

Ejercicio 3.

Realizar un programa en el que 3 robots realicen escaleras de 4 escalones. El tamaño de cada escalón se incrementa en un 1 respecto al escalón anterior. El primer escalón será de 1x1, el segundo de 2x2 y así sucesivamente, como se muestra a continuación:



Al finalizar el recorrido, cada robot debe informar la cantidad de escalones en los que la cantidad de papeles superó en 1 a la cantidad de flores. Las esquinas deben quedar sin modificar.

programa TP11_E3

procesos

proceso contarFloresEsquina (ES flores: numero)

variables

flores_esq: numero

comenzar

flores_esq:=0

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

flores_esq:=flores_esq+1

repetir flores_esq

depositarFlor

fin

proceso contarPapelesEsquina (ES papeles: numero)

variables

papeles_esq: numero

comenzar

papeles_esq:=0

mientras (HayPapelEnLaEsquina)

```

    tomarPapel
    papeles:=papeles+1
    papeles_esq:=papeles_esq+1
    repetir papeles_esq
    depositarPapel
fin
proceso escalon (ES flores: numero; ES papeles: numero; E pasos: numero)
comenzar
    repetir 2
        repetir pasos
            contarFloresEsquina(flores)
            contarPapelesEsquina(papeles)
        mover
        derecha
    repetir 2
        derecha
fin
proceso escalera (ES escalones: numero)
variables
    flores, papeles, pasos: numero
comenzar
    pasos:=1
    repetir 4
        flores:=0
        papeles:=0
        escalon(flores,papeles,pasos)
        pasos:=pasos+1
        si (papeles-flores=1)
            escalones:=escalones+1
fin
areas
ciudad: AreaC(1,1,100,100)
robots
robot recolector
variables
    escalones: numero
comenzar
    escalones:=0
    escalera(escalones)
    Informar ('EscalonesConUnPapelMás',escalones)
fin
variables
robot1, robot2, robot3: recolector
comenzar
    AsignarArea(robot1,ciudad)
    AsignarArea(robot2,ciudad)
    AsignarArea(robot3,ciudad)
    Iniciar(robot1,12,14)
    Iniciar(robot2,17,10)
    Iniciar(robot3,22,6)

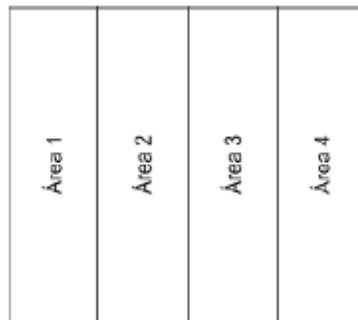
```

fin

Ejercicio 4.

Realizar un programa en el que dos robots se encargan de limpiar la ciudad. La ciudad se dividió en 4 áreas: las impares (1 y 3) deben limpiarse de flores; y las pares (2 y 4) deben limpiarse de papeles. Un robot debe encargarse de las áreas impares y otro robot de las pares. Modularizar el recorrido de cada área:

- Área 1: desde la avenida 1 hasta la avenida 25.
- Área 2: desde la avenida 26 hasta la avenida 50.
- Área 3: desde la avenida 51 hasta la avenida 75.
- Área 4: desde la avenida 76 hasta la avenida 100.



Analizar (no es necesario implementar) qué se debería modificar si, ahora, se pide que la ciudad se divida en 20 áreas:

- Área 1: Avenidas 1 a 5.
- Área 2: Avenidas 6 a 10.
- ...
- Área 19: Avenidas 91 a 95.
- Área 20: Avenidas 96 a 100.

programa TP11_E4

procesos

proceso juntarFloresEsquina

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

fin

proceso juntarPapelesEsquina

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

fin

proceso recorrerAreaImpar

comenzar

repetir 24

repetir 99

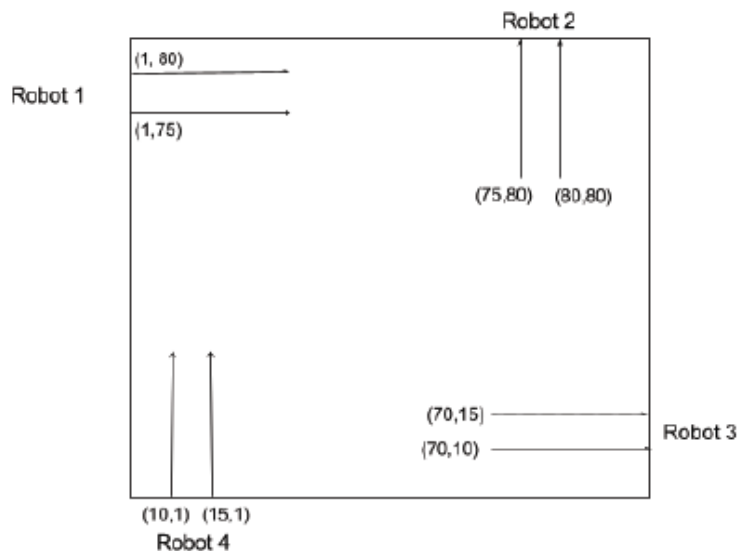
juntarFloresEsquina

```
    mover
    juntarFloresEsquina
    Pos (PosAv+1,1)
repetir 99
    juntarFloresEsquina
    mover
    juntarFloresEsquina
fin
proceso recorrerAreaPar
comenzar
    repetir 24
        repetir 99
            juntarPapelesEsquina
            mover
            juntarPapelesEsquina
            Pos (PosAv+1,1)
        repetir 99
            juntarPapelesEsquina
            mover
            juntarPapelesEsquina
fin
areas
area1: AreaP(1,1,25,100)
area2: AreaP(26,1,50,100)
area3: AreaP(51,1,75,100)
area4: AreaP(76,1,100,100)
robots
robot limpiador1
comenzar
    recorrerAreaImpar
    Pos (PosAv+26,1)
    recorrerAreaImpar
fin
robot limpiador2
comenzar
    recorrerAreaPar
    Pos (PosAv+26,1)
    recorrerAreaPar
fin
variables
robot1: limpiador1
robot2: limpiador2
comenzar
    AsignarArea(robot1,area1)
    AsignarArea(robot1,area3)
    AsignarArea(robot2,area2)
    AsignarArea(robot2,area4)
    Iniciar(robot1,1,1)
    Iniciar(robot2,26,1)
fin
```

Ejercicio 5.

Realizar un programa en el que cuatro robots realizan las siguientes actividades:

- El robot 1 debe limpiar de flores las primeras 15 esquinas de las calles 75 y 80. Al finalizar cada calle, debe depositar todas las flores en la última esquina.
- El robot 2 debe limpiar de papeles las últimas 20 esquinas de las avenidas 75 y 80. Al finalizar cada avenida, debe depositar todos los papeles en la primera esquina.
- El robot 3 debe limpiar de flores las últimas 30 esquinas de las calles 10 y 15. Al finalizar cada calle, debe depositar todas las flores en la última esquina.
- El robot 4 debe limpiar de papeles las primeras 10 esquinas de las avenidas 10 y 15. Al finalizar cada avenida, debe depositar todos los papeles en la primera esquina.



programa TP11_E5

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso depositarFlores (E flores: numero)

comenzar

repetir flores

depositarFlor

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel


```
papeles:=papeles+1
fin
proceso depositarPapeles (E papeles: numero)
comenzar
  repetir papeles
    depositarPapel
```

```
fin
proceso recorrerCalle (E cant: numero)
```

```
variables
  flores: numero
comenzar
  flores:=0
  repetir cant-1
    juntarFloresEsquina(flores)
    mover
    juntarFloresEsquina(flores)
    depositarFlores(flores)
```

```
fin
proceso recorrerAvenida (E cant: numero)
```

```
variables
  av, ca, papeles: numero
comenzar
  av:=PosAv
  ca:=PosCa
  papeles:=0
  repetir cant-1
    juntarPapelesEsquina(papeles)
    mover
    juntarPapelesEsquina(papeles)
    Pos (av,ca)
    depositarPapeles(papeles)
```

```
fin
```

areas

```
area1a: AreaP(1,75,15,75)
area1b: AreaP(1,80,15,80)
area2a: AreaP(75,81,75,100)
area2b: AreaP(80,81,80,100)
area3a: AreaP(71,10,100,10)
area3b: AreaP(71,15,100,15)
area4a: AreaP(10,1,10,10)
area4b: AreaP(15,1,15,10)
```

robots

```
robot limpiador1
variables
  av: numero
comenzar
  av:=PosAv
  derecha
  recorrerCalle(15)
  Pos (av,PosCa+5)
```

```
    recorrerCalle(15)
fin
robot limpiador2
variables
    ca: numero
comenzar
    ca:=PosCa
    recorrerAvenida(20)
    Pos (PosAv+5,ca)
    recorrerAvenida(20)
fin
robot limpiador3
variables
    av: numero
comenzar
    av:=PosAv
    derecha
    recorrerCalle(30)
    Pos (av,PosCa+5)
    recorrerCalle(30)
fin
robot limpiador4
variables
    ca: numero
comenzar
    ca:=PosCa
    recorrerAvenida(10)
    Pos (PosAv+5,ca)
    recorrerAvenida(10)
fin
variables
robot1: limpiador1
robot2: limpiador2
robot3: limpiador3
robot4: limpiador4
comenzar
    AsignarArea(robot1,area1a)
    AsignarArea(robot1,area1b)
    AsignarArea(robot2,area2a)
    AsignarArea(robot2,area2b)
    AsignarArea(robot3,area3a)
    AsignarArea(robot3,area3b)
    AsignarArea(robot4,area4a)
    AsignarArea(robot4,area4b)
    Iniciar(robot1,1,75)
    Iniciar(robot2,75,81)
    Iniciar(robot3,71,10)
    Iniciar(robot4,10,1)
fin
```

Trabajo Práctico N° 12:

Módulo Concurrente (Realizar programas en R-info con distintos tipos de robots. Utilizar el pasaje de mensajes para la comunicación entre robots. Usar la función random para generar valores aleatorios).

Ejercicio 1.

(a) *Dos robots compiten para ver cuál junta más flores. El primer robot recoge todas las flores de la avenida 1, entre las calles 1 y 10. El segundo robot recoge todas las flores de la avenida 2, entre las calles 11 y 20. Al finalizar el recorrido, el robot que recogió mayor cantidad de flores debe informar la diferencia de flores que obtuvo respecto al robot perdedor (el que obtuvo menos flores). Los robots inician en las esquinas (1,1) y (2,11), respectivamente.*

programa TP12_E1a

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso recorrerAvenida (ES flores: numero)

comenzar

repetir 9

juntarFloresEsquina(flores)

mover

juntarFloresEsquina(flores)

fin

areas

area1: AreaP(1,1,1,10)

area2: AreaP(2,11,2,20)

robots

robot jugador1

variables

flores1, flores2: numero

comenzar

flores1:=0

recorrerAvenida(flores1)

EnviarMensaje(flores1,robot2)

RecibirMensaje(flores2,robot2)

si (flores1>flores2)

Informar ('DiferenciaRobotGanador',flores1-flores2)

fin

robot jugador2

variables

```

    flores1, flores2: numero
comenzar
    flores2:=0
    recorrerAvenida(flores2)
    EnviarMensaje(flores2,robot1)
    RecibirMensaje(flores1,robot1)
    si (flores2>flores1)
        Informar ('DiferenciaRobotGanador',flores2-flores1)
fin
variables
    robot1: jugador1
    robot2: jugador2
comenzar
    AsignarArea(robot1,area1)
    AsignarArea(robot2,area2)
    Iniciar(robot1,1,1)
    Iniciar(robot2,2,1)
fin

```

(b) Modificar el ejercicio anterior, considerando que, ahora, habrá un robot fiscalizador, que será responsable de informar la diferencia de flores que obtuvo el ganador con respecto al perdedor. El robot fiscalizador se ubica en la esquina (2,1).

```

programa TP12_E1b
procesos
    proceso juntarFloresEsquina (ES flores: numero)
comenzar
    mientras (HayFlorEnLaEsquina)
        tomarFlor
        flores:=flores+1
fin
    proceso recorrerAvenida (ES flores: numero)
comenzar
    repetir 9
        juntarFloresEsquina(flores)
        mover
        juntarFloresEsquina(flores)
fin
areas
    area1: AreaP(1,1,1,10)
    area2: AreaP(2,1,2,20)
    areaF: AreaP(2,1,2,1)
robots
    robot jugador
variables
    rob, flores: numero
comenzar
    flores:=0

```

```

    RecibirMensaje(rob,robotF)
    recorrerAvenida(flores)
    EnviarMensaje(rob,robotF)
    EnviarMensaje(flores,robotF)
fin
robot fiscalizador
variables
    rob, flores1, flores2: numero
comenzar
    EnviarMensaje(1,robot1)
    EnviarMensaje(2,robot2)
    repetir 2
        RecibirMensaje(rob,*)
        si (rob=1)
            RecibirMensaje(flores1,robot1)
        sino
            RecibirMensaje(flores2,robot2)
    si (flores1>flores2)
        Informar ('Robot1 GanadorConDiferencia',flores1-flores2)
    sino
        si (flores2>flores1)
            Informar ('Robot2 GanadorConDiferencia',flores2-flores1)
        sino
            Informar ('AmbosRobotsRecogieron',flores1)
fin
variables
    robot1, robot2: jugador
    robotF: fiscalizador
comenzar
    AsignarArea(robot1,area1)
    AsignarArea(robot2,area2)
    AsignarArea(robotF,areaF)
    Iniciar(robot1,1,1)
    Iniciar(robot2,2,1)
    Iniciar(robotF,2,1)
fin

```

(c) *Modificar el ejercicio anterior para que, ahora, participen 6 robots.*

- *Robot 1: Avenida 1, entre las calles 1 y 10.*
- *Robot 2: Avenida 2, entre las calles 11 y 20.*
- *Robot 3: Avenida 3, entre las calles 21 y 30.*
- *Robot 4: Avenida 4, entre las calles 31 y 40.*
- *Robot 5: Avenida 5, entre las calles 41 y 50.*
- *Robot 6: Avenida 6, entre las calles 51 y 60.*
- *Fiscalizador: Avenida 2, calle 1.*

El fiscalizador deberá informar la cantidad de flores que juntó el robot ganador.

programa TP12_E1c**procesos****proceso** juntarFloresEsquina (ES flores: numero)**comenzar**

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin**proceso** recorrerAvenida (ES flores: numero)**comenzar**

repetir 9

juntarFloresEsquina(flores)

mover

juntarFloresEsquina(flores)

fin**proceso** actualizarMaximo (E flores: numero; ES floresMax: numero)**comenzar**

si (flores>floresMax)

floresMax:=flores

fin**areas**

area1: AreaP(1,1,1,10)

area2: AreaP(2,11,2,20)

area3: AreaP(3,21,3,30)

area4: AreaP(4,31,4,40)

area5: AreaP(5,41,5,50)

area6: AreaP(6,51,6,60)

areaF: AreaP(2,1,2,1)

robots

robot jugador

variables

flores: numero

comenzar

flores:=0

recorrerAvenida(flores)

EnviarMensaje(flores,robotF)

fin

robot fiscalizador

variables

flores, floresMax: numero

comenzar

floresMax:=0

repetir 6

RecibirMensaje(flores,*)

actualizarMaximo(flores,floresMax)

Informar ('FloresGanador',floresMax)

fin**variables**

robot1, robot2, robot3, robot4, robot5, robot6: jugador

robotF: fiscalizador

comenzar

AsignarArea(robot1,area1)

AsignarArea(robot2,area2)

AsignarArea(robot3,area3)

AsignarArea(robot4,area4)

AsignarArea(robot5,area5)

AsignarArea(robot6,area6)

AsignarArea(robotF,areaF)

Iniciar(robot1,1,1)

Iniciar(robot2,2,11)

Iniciar(robot3,3,21)

Iniciar(robot4,4,31)

Iniciar(robot5,5,41)

Iniciar(robot6,6,51)

Iniciar(robotF,2,1)

fin

(d) Modificar el ejercicio anterior para que, ahora, el fiscalizador informe también cuál fue el robot ganador.

programa TP12_E1d

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso recorrerAvenida (ES flores: numero)

comenzar

repetir 9

juntarFloresEsquina(flores)

mover

juntarFloresEsquina(flores)

fin

proceso actualizarMaximos (E flores: numero; E rob: numero; ES floresMax: numero; ES robMax: numero)

comenzar

si (flores>floresMax)

floresMax:=flores

robMax:=rob

fin

areas

area1: AreaP(1,1,1,10)

area2: AreaP(2,11,2,20)

area3: AreaP(3,21,3,30)

area4: AreaP(4,31,4,40)

area5: AreaP(5,41,5,50)

area6: AreaP(6,51,6,60)

areaF: AreaP(2,1,2,1)

robots

robot jugador

variables

rob, flores: numero

comenzar

flores:=0

RecibirMensaje(rob,robotF)

recorrerAvenida(flores)

EnviarMensaje(rob,robotF)

EnviarMensaje(flores,robotF)

fin

robot fiscalizador

variables

rob, robMax, flores, floresMax: numero

comenzar

floresMax:=0

robMax:=0

EnviarMensaje(1,robot1)

EnviarMensaje(2,robot2)

EnviarMensaje(3,robot3)

EnviarMensaje(4,robot4)

EnviarMensaje(5,robot5)

EnviarMensaje(6,robot6)

repetir 6

RecibirMensaje(rob,*)

si (rob=1)

RecibirMensaje(flores,robot1)

sino

si (rob=2)

RecibirMensaje(flores,robot2)

sino

si (rob=3)

RecibirMensaje(flores,robot3)

sino

si (rob=4)

RecibirMensaje(flores,robot4)

sino

si (rob=5)

RecibirMensaje(flores,robot5)

sino

RecibirMensaje(flores,robot6)

actualizarMaximos(flores,rob,floresMax,robMax)

Informar ('RobotGanador_Flores',robMax,floresMax)

fin

variables

robot1, robot2, robot3, robot4, robot5, robot6: jugador

robotF: fiscalizador

comenzar

```
AsignarArea(robot1,area1)
AsignarArea(robot2,area2)
AsignarArea(robot3,area3)
AsignarArea(robot4,area4)
AsignarArea(robot5,area5)
AsignarArea(robot6,area6)
AsignarArea(robotF,areaF)
Iniciar(robot1,1,1)
Iniciar(robot2,2,11)
Iniciar(robot3,3,21)
Iniciar(robot4,4,31)
Iniciar(robot5,5,41)
Iniciar(robot6,6,51)
Iniciar(robotF,2,1)
```

fin

(e) Analizar (no es necesario implementar): ¿Cómo se puede implementar el inciso (c) sin robot fiscalizador? ¿Qué cantidad de robots participarán del juego? ¿Qué cantidad de mensajes deben enviarse?

En el inciso (c), sin robot fiscalizador, participarán 6 robots en el juego y cada uno deberá enviar y recibir 5 mensajes, uno por cada uno de los 5 robots restantes.

Ejercicio 2.

Realizar un programa en el que 3 robots realizan una escalera de 4 escalones cada uno. Todos los escalones tienen un ancho fijo de 1 y un alto aleatorio entre 1 y 5. Al finalizar el recorrido, cada robot deberá enviar al robot jefe la cantidad de escalones que tenían más flores que papeles. Una vez que los tres robots finalizaron, el robot jefe deberá informar la suma de las cantidades enviadas por los 3 robots.

- El robot jefe inicia en la esquina (1,1).
- El robot 1 inicia en la esquina (2,1).
- El robot 2 inicia en la esquina (7,1).
- El robot 3 inicia en la esquina (12,1).

programa TP12_E2

procesos

proceso izquierda

comenzar

repetir 3

derecha

fin

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

proceso juntarFloresyPapelesEsquina (ES flores: numero; ES papeles: numero)

comenzar

juntarFloresEsquina(flores)

juntarPapelesEsquina(papeles)

fin

proceso escalon (E alto: numero; ES escalones: numero)

variables

flores, papeles: numero

comenzar

flores:=0

papeles:=0

repetir alto

juntarFloresyPapelesEsquina(flores,papeles)

mover

juntarFloresyPapelesEsquina(flores,papeles)

derecha

mover

```
    izquierda
    si (flores>papeles)
        escalones:=escalones+1
fin
areas
area1: AreaP(2,1,6,21)
area2: AreaP(7,1,11,21)
area3: AreaP(12,1,16,21)
areaJ: AreaP(1,1,1,1)
robots
robot recogedor
variables
    alto, escalones: numero
comenzar
    escalones:=0
    repetir 4
        Random(alto,1,5)
        escalon(alto,escalones)
        EnviarMensaje(escalones,robotJ)
fin
robot jefe
variables
    sumaEscalones, escalones: numero
comenzar
    sumaEscalones:=0
    repetir 3
        RecibirMensaje(escalones,*)
        sumaEscalones:=sumaEscalones+escalones
        Informar ('SumaEscalones',sumaEscalones)
fin
variables
    robot1, robot2, robot3: recogedor
    robotJ: jefe
comenzar
    AsignarArea(robot1,area1)
    AsignarArea(robot2,area2)
    AsignarArea(robot3,area3)
    AsignarArea(robotJ,areaJ)
    Iniciar(robot1,2,1)
    Iniciar(robot2,7,1)
    Iniciar(robot3,12,1)
    Iniciar(robotJ,1,1)
fin
```

Ejercicio 3.

(a) Realizar un programa con 2 equipos:

- El equipo A, compuesto por los robots A1 y A2, debe juntar papeles de las primeras 20 esquinas de la calle 1.
- El equipo B, compuesto por los robots B1 y B2, debe juntar flores de las primeras 20 esquinas de la calle 5.

Los robots A1 y B1 deberán realizar las 10 primeras esquinas de su recorrido y, al finalizar, avisarán a los robots A2 y B2, respectivamente, para que continúen con las siguientes 10 esquinas. El segundo robot de cada equipo debe informar la cantidad de elementos recogidos en las 20 esquinas. Inicializar los 4 robots en las esquinas que se considere más apropiadas según el trayecto que le corresponde realizar a cada uno.

programa TP12_E3a

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

proceso juntarFlores (ES flores: numero)

comenzar

repetir 9

juntarFloresEsquina(flores)

mover

juntarFloresEsquina(flores)

fin

proceso juntarPapeles (ES papeles: numero)

comenzar

repetir 9

juntarPapelesEsquina(papeles)

mover

juntarPapelesEsquina(papeles)

fin

areas

areaA1: AreaP(1,1,10,1)

areaA2: AreaP(11,1,20,1)

areaB1: AreaP(1,5,10,5)

areaB2: AreaP(11,5,20,5)

robots

```
robot jugadorA1
variables
  papeles: numero
comenzar
  papeles:=0
  derecha
  juntarPapeles(papeles)
  EnviarMensaje(papeles,robotA2)
fin
robot jugadorA2
variables
  papeles: numero
comenzar
  RecibirMensaje(papeles,robotA1)
  derecha
  juntarPapeles(papeles)
  Informar ('TotalPapelesEquipoA',papeles)
fin
robot jugadorB1
variables
  flores: numero
comenzar
  flores:=0
  derecha
  juntarFlores(flores)
  EnviarMensaje(flores,robotB2)
fin
robot jugadorB2
variables
  flores: numero
comenzar
  RecibirMensaje(flores,robotB1)
  derecha
  juntarFlores(flores)
  Informar ('TotalFloresEquipoB',flores)
fin
variables
robotA1: jugadorA1
robotA2: jugadorA2
robotB1: jugadorB1
robotB2: jugadorB2
comenzar
  AsignarArea(robotA1,areaA1)
  AsignarArea(robotA2,areaA2)
  AsignarArea(robotB1,areaB1)
  AsignarArea(robotB2,areaB2)
  Iniciar(robotA1,1,1)
  Iniciar(robotA2,11,1)
  Iniciar(robotB1,1,5)
  Iniciar(robotB2,11,5)
```

fin

(b) *Modificar el ejercicio anterior para que cada equipo repita el recorrido con las siguientes 20 esquinas de sus correspondientes calles.*

programa TP12_E3b

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

proceso juntarFlores (ES flores: numero)

comenzar

repetir 9

juntarFloresEsquina(flores)

mover

juntarFloresEsquina(flores)

fin

proceso juntarPapeles (ES papeles: numero)

comenzar

repetir 9

juntarPapelesEsquina(papeles)

mover

juntarPapelesEsquina(papeles)

fin

areas

areaA1a: AreaP(1,1,10,1)

areaA1b: AreaP(21,1,30,1)

areaA2a: AreaP(11,1,20,1)

areaA2b: AreaP(31,1,40,1)

areaB1a: AreaP(1,5,10,5)

areaB1b: AreaP(21,5,30,5)

areaB2a: AreaP(11,5,20,5)

areaB2b: AreaP(31,5,40,5)

robots

robot jugadorA1

variables

papeles: numero

comenzar

papeles:=0

```
derecha
juntarPapeles(papeles)
EnviarMensaje(papeles,robotA2)
Pos (PosAv+11,PosCa)
RecibirMensaje(papeles,robotA2)
juntarPapeles(papeles)
EnviarMensaje(papeles,robotA2)
fin
robot jugadorA2
variables
papeles: numero
comenzar
RecibirMensaje(papeles,robotA1)
derecha
juntarPapeles(papeles)
EnviarMensaje(papeles,robotA1)
Pos (PosAv+11,PosCa)
RecibirMensaje(papeles,robotA1)
juntarPapeles(papeles)
Informar ('TotalPapelesEquipoA',papeles)
fin
robot jugadorB1
variables
flores: numero
comenzar
flores:=0
derecha
juntarFlores(flores)
EnviarMensaje(flores,robotB2)
Pos (PosAv+11,PosCa)
RecibirMensaje(flores,robotB2)
juntarFlores(flores)
EnviarMensaje(flores,robotB2)
fin
robot jugadorB2
variables
flores: numero
comenzar
RecibirMensaje(flores,robotB1)
derecha
juntarFlores(flores)
EnviarMensaje(flores,robotB1)
Pos (PosAv+11,PosCa)
RecibirMensaje(flores,robotB1)
juntarPapeles(flores)
Informar ('TotalPapelesEquipoB',flores)
fin
variables
robotA1: jugadorA1
robotA2: jugadorA2
```

robotB1: jugadorB1

robotB2: jugadorB2

comenzar

AsignarArea(robotA1,areaA1a)

AsignarArea(robotA1,areaA1b)

AsignarArea(robotA2,areaA2a)

AsignarArea(robotA2,areaA2b)

AsignarArea(robotB1,areaB1a)

AsignarArea(robotB1,areaB1b)

AsignarArea(robotB2,areaB2a)

AsignarArea(robotB2,areaB2b)

Iniciar(robotA1,1,1)

Iniciar(robotA2,11,1)

Iniciar(robotB1,1,5)

Iniciar(robotB2,11,5)

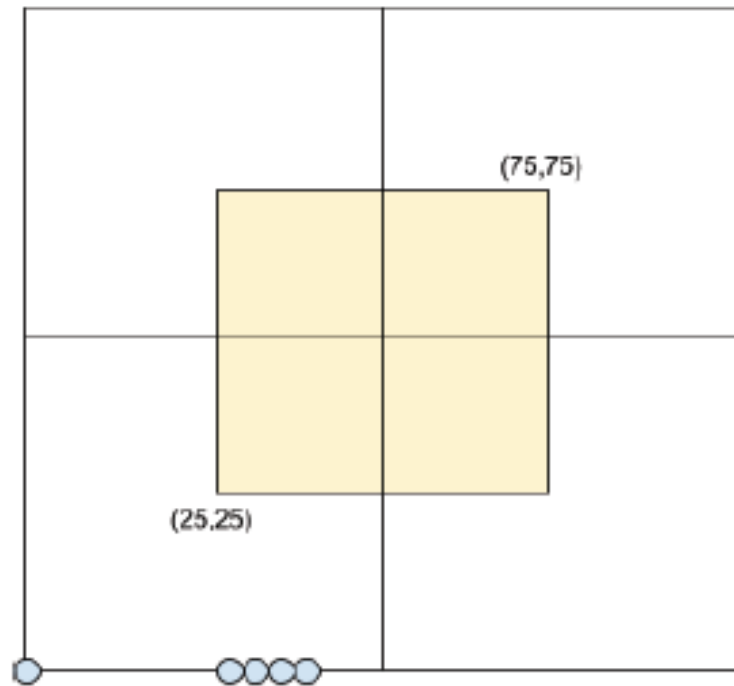
fin

(c) Analizar (no es necesario implementar) cómo se implementaría el inciso (b) si, ahora, cada equipo debe realizar 5 segmentos de 20 esquinas.

El inciso (b), si, ahora, cada equipo debe realizar 5 segmentos de 20 esquinas se implementaría de la misma manera, pero el primer robot de cada equipo debería enviar 5 y recibir 4 mensajes y el segundo robot de cada equipo debería enviar 4 y recibir 5 mensajes.

Ejercicio 4.

Realizar un programa en el que un robot fiscalizador controla el acceso de 4 robots recolectores al cuadrante encerrado entre las esquinas (25,25) y (75,75). Para ello, el robot fiscalizador avisa a un robot recolector aleatorio que puede ingresar al área. El robot que recibe la autorización de acceso calcula una esquina aleatoria dentro del área, limpia dicha esquina de flores y papeles, regresa a su esquina y avisa al robot fiscalizador que ya finalizó.



Se realizarán, en total, 10 accesos al cuadrante entre los 4 robots recolectores. Al finalizar, el robot fiscalizador deberá indicar al robot ganador que se posicione en la esquina (50,50). El robot fiscalizador inicia en la esquina (1,1) y los robots recolectores inician en las esquinas (25,1), (30,1), (35,1) y (40,1), respectivamente.

programa TP12_E4

procesos

proceso juntarFloresyPapelesEsquina (ES cant: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

cant:=cant+1

mientras (HayPapelEnLaEsquina)

tomarPapel

cant:=cant+1

fin

proceso acceder (ES cant: numero)

variables

avIni, caIni, av, ca: numero

comenzar

```

avIni:=PosAv
caIni:=PosCa
Random(av,25,75)
Random(ca,25,75)
Pos (av,ca)
juntarFloresyPapelesEsquina(cant)
Pos (avIni,caIni)
fin
proceso obtenerMaximo (E cant1: numero; E cant2: numero; E cant3: numero; E cant4:
numero; ES robMax: numero)
variables
cantMax: numero
comenzar
robMax:=-1
si (cant1>cantMax)
cantMax:=cant1
robMax:=1
si (cant2>cantMax)
cantMax:=cant2
robMax:=2
si (cant3>cantMax)
cantMax:=cant3
robMax:=3
si (cant4>cantMax)
cantMax:=cant4
robMax:=4
fin
proceso avisarRobots (E robMax: numero)
variables
rob: numero
gane: boolean
comenzar
rob:=1
repetir 4
gane:=F
si (rob=robMax)
gane:=V
si (rob=1)
EnviarMensaje(gane,robot1)
sino
si (rob=2)
EnviarMensaje(gane,robot2)
sino
si (rob=3)
EnviarMensaje(gane,robot3)
sino
EnviarMensaje(gane,robot4)
rob:=rob+1
fin
areas

```

areaPC: AreaPC(25,25,75,75)

area1: AreaP(25,1,25,1)

area2: AreaP(30,1,30,1)

area3: AreaP(35,1,35,1)

area4: AreaP(40,1,40,1)

areaF: AreaP(1,1,1,1)

robots

robot recolector

variables

cant: numero

ok, gane: boolean

comenzar

RecibirMensaje(ok,robotF)

mientras (ok)

 RecibirMensaje(cant,robotF)

 acceder(cant)

 EnviarMensaje(cant,robotF)

 RecibirMensaje(ok,robotF)

RecibirMensaje(gane,robotF)

si (gane)

 Pos (50,50)

fin

robot fiscalizador

variables

rob, robMax, cant1, cant2, cant3, cant4: numero

ok: boolean

comenzar

cant1:=0

cant2:=0

cant3:=0

cant4:=0

ok:=V

repetir 10

 Random(rob,1,4)

 si (rob=1)

 EnviarMensaje(ok,robot1)

 EnviarMensaje(cant1,robot1)

 RecibirMensaje(cant1,robot1)

 sino

 si (rob=2)

 EnviarMensaje(ok,robot2)

 EnviarMensaje(cant2,robot2)

 RecibirMensaje(cant2,robot2)

 sino

 si (rob=3)

 EnviarMensaje(ok,robot3)

 EnviarMensaje(cant3,robot3)

 RecibirMensaje(cant3,robot3)

 sino

 si (rob=4)

```
    EnviarMensaje(ok,robot4)
    EnviarMensaje(cant4,robot4)
    RecibirMensaje(cant4,robot4)
ok:=F
EnviarMensaje(ok,robot1)
EnviarMensaje(ok,robot2)
EnviarMensaje(ok,robot3)
EnviarMensaje(ok,robot4)
obtenerMaximo(cant1,cant2,cant3,cant4,robMax)
avisarRobots(robMax)
Informar ('RobotGanador',robMax)
```

fin

variables

```
robot1, robot2, robot3, robot4: recolector
robotF: fiscalizador
```

comenzar

```
AsignarArea(robot1,areaPC)
AsignarArea(robot1,area1)
AsignarArea(robot2, areaPC)
AsignarArea(robot2,area2)
AsignarArea(robot3,areaPC)
AsignarArea(robot3,area3)
AsignarArea(robot4,areaPC)
AsignarArea(robot4,area4)
AsignarArea(robotF,areaF)
Iniciar(robot1,25,1)
Iniciar(robot2,30,1)
Iniciar(robot3,35,1)
Iniciar(robot4,40,1)
Iniciar(robotF,1,1)
```

fin

Trabajo Práctico N° 13:

Módulo Concurrente (Realizar programas en R-info con distintos tipos de robots. Utilizar memoria compartida para la comunicación y sincronización entre robots. Combinar problemas con memoria compartida y pasaje de mensajes).

Ejercicio 1.

Realizar un programa con 2 robots recolectores de flores (floreros) y 2 robots recolectores de papeles (papeleros). Los floreros comparten área y tienen 5 intentos cada uno para juntar las flores de una esquina dentro de dicha área, elegida al azar en cada intento. Del mismo modo, los papeleros comparten área y tienen 3 intentos cada uno para juntar los papeles. En cada intento, cada robot va a la esquina al azar, junta todos los elementos (flores o papeles, según le corresponda) y vuelve a su esquina original. Al finalizar sus intentos, cada robot debe acceder a la esquina (10,10) y depositar los elementos recogidos de a uno.

- Área de floreros: (1,1) a (5,10).
- Área de papeleros: (6,1) a (10,9).
- Esquinas de inicio de floreros: (6,10) y (7,10).
- Esquinas de inicio de papeleros: (8,10) y (9,10).

programa TP13_E1

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

areas

areaC: AreaC(10,10,10,10)

areaF: AreaPC(1,1,5,10)

areaP: AreaPC(6,1,10,9)

areaF1: AreaP(6,10,6,10)

areaF2: AreaP(7,10,7,10)

areaP1: AreaP(8,10,8,10)

areaP2: AreaP(9,10,9,10)

robots

robot recolectorFlores

variables

avIni, caIni, av, ca, flores: numero

comenzar

avIni:=PosAv

caIni:=PosCa

flores:=0

repetir 5

Random(av,1,5)

Random(ca,1,10)

BloquearEsquina(av,ca)

Pos (av,ca)

juntarFloresEsquina(flores)

Pos (avIni,caIni)

LiberarEsquina(av,ca)

repetir flores

BloquearEsquina(10,10)

Pos (10,10)

depositarFlor

Pos (avIni,caIni)

LiberarEsquina(10,10)

fin

robot recolectorPapeles

variables

avIni, caIni, av, ca, papeles: numero

comenzar

avIni:=PosAv

caIni:=PosCa

papeles:=0

repetir 3

Random(av,6,10)

Random(ca,1,9)

BloquearEsquina(av,ca)

Pos (av,ca)

juntarPapelesEsquina(papeles)

Pos (avIni,caIni)

LiberarEsquina(av,ca)

repetir papeles

BloquearEsquina(10,10)

Pos (10,10)

depositarPapel

Pos (avIni,caIni)

LiberarEsquina(10,10)

fin**variables**

robotF1, robotF2: recolectorFlores

robotP3, robotP4: recolectorPapeles

comenzar

AsignarArea(robotF1,areaC)

AsignarArea(robotF1,areaF)

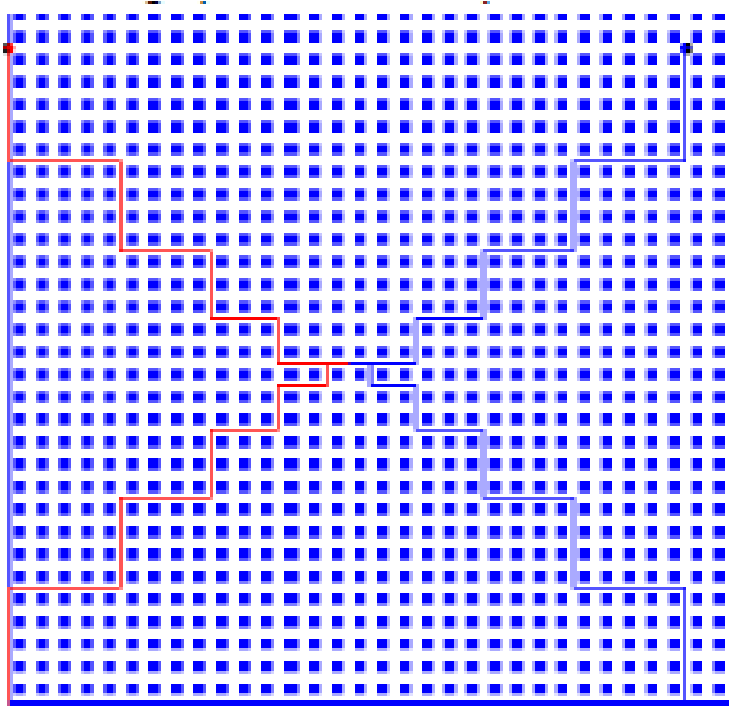
AsignarArea(robotF1,areaF1)

```
AsignarArea(robotF2,areaC)
AsignarArea(robotF2,areaF)
AsignarArea(robotF2,areaF2)
AsignarArea(robotP3,areaC)
AsignarArea(robotP3,areaP)
AsignarArea(robotP3,areaP1)
AsignarArea(robotP4,areaC)
AsignarArea(robotP4,areaP)
AsignarArea(robotP4,areaP2)
Iniciar(robotF1,6,10)
Iniciar(robotF2,7,10)
Iniciar(robotP3,8,10)
Iniciar(robotP4,9,10)
```

fin

Ejercicio 2.

Realizar un programa en el cual 2 robots corren una carrera. El recorrido realizado por cada uno es el que se muestra en la siguiente figura. Durante el recorrido, el robot 1 debe juntar todas las flores que encuentre en los vértices de cada escalón, mientras que el robot 2 debe juntar todos los papeles que encuentre en los vértices de cada escalón. Al finalizar, deben informar la cantidad de elementos recogidos. El robot 1 debe iniciar su recorrido en la esquina (1,1) y el robot 2 debe iniciar su recorrido en la esquina (31,1). Al finalizar la carrera, un robot jefe (inicializado en la esquina (15,1)) debe informar qué robot llegó primero a la esquina central de su recorrido.



programa TP13_E2

procesos

proceso izquierda

comenzar

repetir 3

derecha

fin

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

proceso escaleraDecF (ES flores: numero)

variables

pasos: numero

comenzar

pasos:=5

juntarFloresEsquina(flores)

repetir 4

repetir pasos

mover

juntarFloresEsquina(flores)

derecha

repetir pasos

mover

juntarFloresEsquina(flores)

izquierda

pasos:=pasos-1

fin

proceso escalonCompF (ES flores: numero; E rob: numero)

comenzar

mover

juntarFloresEsquina(flores)

derecha

BloquearEsquina(16,16)

mover

EnviarMensaje(rob,robotJ)

juntarFloresEsquina(flores)

repetir 2

derecha

mover

LiberarEsquina(16,16)

fin

proceso escaleraCrecF (ES flores: numero)

variables

pasos: numero

comenzar

pasos:=2

repetir 4

repetir pasos

mover

juntarFloresEsquina(flores)

derecha

repetir pasos

mover

juntarFloresEsquina(flores)

izquierda

pasos:=pasos+1

fin

proceso escaleraDecP (ES papeles: numero)

variables

```
pasos: numero
comenzar
pasos:=5
juntarPapelesEsquina(papeles)
repetir 4
  repetir pasos
  mover
juntarPapelesEsquina(papeles)
izquierda
repetir pasos
  mover
juntarPapelesEsquina(papeles)
derecha
pasos:=pasos-1
fin
proceso escalonCompP (ES papeles: numero; E rob: numero)
comenzar
  mover
  juntarPapelesEsquina(papeles)
  izquierda
  BloquearEsquina(16,16)
  mover
  EnviarMensaje(rob,robotJ)
  juntarPapelesEsquina(papeles)
  repetir 2
    derecha
  mover
  LiberarEsquina(16,16)
fin
proceso escaleraCrecP (ES papeles: numero)
variables
pasos: numero
comenzar
pasos:=2
repetir 4
  repetir pasos
  mover
  juntarPapelesEsquina(papeles)
  izquierda
  repetir pasos
  mover
  juntarPapelesEsquina(papeles)
  derecha
  pasos:=pasos+1
z fin
areas
areaPC: AreaPC(16,16,16,16)
areaF: AreaPC(1,1,15,30)
areaP: AreaP(17,1,31,30)
robots
```

robot jugador1

variables

rob, flores: numero

comenzar

flores:=0

RecibirMensaje(rob,robotJ)

escaleraDecF(flores)

escalonCompF(flores,rob)

escaleraCrecF(flores)

Informar ('Flores',flores)

EnviarMensaje(V,robotJ)

fin

robot jugador2

variables

rob, papeles: numero

comenzar

papeles:=0

RecibirMensaje(rob,robotJ)

escaleraDecP(papeles)

escalonCompP(papeles,rob)

escaleraCrecP(papeles)

Informar ('Papeles',papeles)

EnviarMensaje(V,robotJ)

fin

robot jefe

variables

rob, robGanador: numero

termino: boolean

comenzar

EnviarMensaje(1,robot1)

EnviarMensaje(2,robot2)

RecibirMensaje(robGanador,*)

RecibirMensaje(rob,*)

repetir 2

RecibirMensaje(termino,*)

Informar ('RobotGanador',robGanador)

fin

variables

robot1: jugador1

robot2: jugador2

robotJ: jefe

comenzar

AsignarArea(robot1,areaPC)

AsignarArea(robot1,areaF)

AsignarArea(robot2,areaPC)

AsignarArea(robot2,areaP)

AsignarArea(robotJ,areaF)

Iniciar(robot1,1,1)

Iniciar(robot2,31,1)

Iniciar(robotJ,15,1)

fin

Ejercicio 3.

Realizar un programa donde 4 robots colaboren para recoger todas las flores de una esquina indicada por un robot jefe, seleccionada, de manera aleatoria, dentro del cuadrante (2,2) y (10,10). Para ello, el jefe determina, inicialmente, una esquina y los robots deben accederla, tomar de a una las flores y volver a su posición inicial. Cuando los robots terminan, el jefe deberá informar cuál de ellos logró recoger más flores. Las esquinas de inicio de los robots deberán ser jefe (1,1) y robots (2,1), (3,1), (4,1) y (5,1).

programa TP13_E3**procesos**

proceso accederEsquina (ES flores: numero; E av: numero; E ca: numero)

variables

avIni, caIni: numero

comenzar

avIni:=PosAv

caIni:=PosCa

BloquearEsquina(av,ca)

Pos (av,ca)

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

Pos (avIni,caIni)

LiberarEsquina(av,ca)

BloquearEsquina(av,ca)

Pos (av,ca)

Pos (avIni,caIni)

LiberarEsquina(av,ca)

fin

proceso actualizarMaximo (E flores: numero; E rob: numero; ES floresMax: numero;

ES robMax: numero)

comenzar

si (flores>floresMax)

floresMax:=flores

robMax:=rob

fin**areas**

areaPC: AreaPC(2,2,10,10)

area1: AreaP(2,1,2,1)

area2: AreaP(3,1,3,1)

area3: AreaP(4,1,4,1)

area4: AreaP(5,1,5,1)

areaJ: AreaP(1,1,1,1)

robots

robot recolector

variables

rob, av, ca, flores: numero

comenzar

flores:=0

```

RecibirMensaje(rob,robotJ)
RecibirMensaje(av,robotJ)
RecibirMensaje(ca,robotJ)
accederEsquina(flores,av,ca)
EnviarMensaje(rob,robotJ)
EnviarMensaje(flores,robotJ)

```

fin

robot jefe

variables

rob, robMax, av, ca, flores, floresMax: numero

comenzar

```

floresMax:=0
robMax:=0
EnviarMensaje(1,robot1)
EnviarMensaje(2,robot2)
EnviarMensaje(3,robot3)
EnviarMensaje(4,robot4)
Random(av,2,10)
Random(ca,2,10)
EnviarMensaje(av,robot1)
EnviarMensaje(ca,robot1)
EnviarMensaje(av,robot2)
EnviarMensaje(ca,robot2)
EnviarMensaje(av,robot3)
EnviarMensaje(ca,robot3)
EnviarMensaje(av,robot4)
EnviarMensaje(ca,robot4)
repetir 4
  RecibirMensaje(rob,*)
  si (rob=1)
    RecibirMensaje(flores,robot1)
  sino
    si (rob=2)
      RecibirMensaje(flores,robot2)
    sino
      si (rob=3)
        RecibirMensaje(flores,robot3)
      sino
        RecibirMensaje(flores,robot4)
  actualizarMaximo(flores,rob,floresMax,robMax)
  Informar ('RobotGanador',robMax)

```

fin

variables

robot1, robot2, robot3, robot4: recolector
robotJ: jefe

comenzar

```

AsignarArea(robot1,areaPC)
AsignarArea(robot1,area1)
AsignarArea(robot2,areaPC)
AsignarArea(robot2,area2)

```

```
AsignarArea(robot3,areaPC)
AsignarArea(robot3,area3)
AsignarArea(robot4,areaPC)
AsignarArea(robot4,area4)
AsignarArea(robotJ,areaJ)
Iniciar(robot1,2,1)
Iniciar(robot2,3,1)
Iniciar(robot3,4,1)
Iniciar(robot4,5,1)
Iniciar(robotJ,1,1)
```

fin

Ejercicio 4.

(a) Realizar un programa en el que 4 robots mueven, de a una, todas las flores de la esquina (10,10) a la esquina (11,11). Para ello, cada robot que toma una flor de la esquina (10,10) la deposita en la esquina (11,11) y, luego, retorna a su esquina inicial. Cada robot que finaliza (o sea que detecta que la esquina (10,10) se ha vaciado) deberá avisar al robot coordinador que ha finalizado. Cuando todos los robots finalizaron, el robot coordinador deberá informar qué robot finalizó último y, a continuación, deberá recolectar todas las flores de la esquina (11,11).

- El robot coordinador inicia en la esquina (1,1).
- Los robots inician en las esquinas (9,9) (9,10) (9,11) y (9,12), respectivamente.

programa TP13_E4a

procesos

proceso trasladarFlor

variables

avIni, caIni: numero

comenzar

avIni:=PosAv

caIni:=PosCa

BloquearEsquina(10,10)

Pos (10,10)

mientras (HayFlorEnLaEsquina)

tomarFlor

BloquearEsquina(11,11)

Pos (11,11)

LiberarEsquina(10,10)

depositarFlor

Pos (avIni,caIni)

LiberarEsquina(11,11)

BloquearEsquina(10,10)

Pos (10,10)

Pos (avIni,caIni)

LiberarEsquina(10,10)

fin

proceso juntarFloresEsquina

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

fin

proceso recolectarFlores

comenzar

Pos (11,11)

juntarFloresEsquina

Pos (1,1)

fin

areas

areaPC1: AreaPC(10,10,10,10)

areaPC2: AreaPC(11,11,11,11)

area1: AreaP(9,9,9,9)

area2: AreaP(9,10,9,10)

area3: AreaP(9,11,9,11)

area4: AreaP(9,12,9,12)

areaC: AreaP(1,1,1,1)

robots

robot recolector

variables

rob: numero

comenzar

RecibirMensaje(rob,robotC)

trasladarFlor

EnviarMensaje(rob,robotC)

fin

robot coordinador

variables

rob: numero

comenzar

EnviarMensaje(1,robot1)

EnviarMensaje(2,robot2)

EnviarMensaje(3,robot3)

EnviarMensaje(4,robot4)

repetir 4

RecibirMensaje(rob,*)

Informar ('RobotFinalizóÚltimo',rob)

recolectarFlores

fin

variables

robot1, robot2, robot3, robot4: recolector

robotC: coordinador

comenzar

AsignarArea(robot1,areaPC1)

AsignarArea(robot1,areaPC2)

AsignarArea(robot1,area1)

AsignarArea(robot2,areaPC1)

AsignarArea(robot2,areaPC2)

AsignarArea(robot2,area2)

AsignarArea(robot3,areaPC1)

AsignarArea(robot3,areaPC2)

AsignarArea(robot3,area3)

AsignarArea(robot4,areaPC1)

AsignarArea(robot4,areaPC2)

AsignarArea(robot4,area4)

AsignarArea(robotC,areaPC2)

AsignarArea(robotC,areaC)

Iniciar(robot1,9,9)

Iniciar(robot2,9,10)

Iniciar(robot3,9,11)

Iniciar(robot4,9,12)

Iniciar(robotC,1,1)

fin

(b) Implementar una variante en la cual los robots, luego de tomar cada flor de la esquina (10,10), vuelvan a su esquina inicial, pasen a la esquina (11,11) a depositarla y, finalmente, vuelvan a la esquina inicial.

programa TP13_E4b

procesos

proceso trasladarFlor

variables

avIni, caIni: numero

comenzar

avIni:=PosAv

caIni:=PosCa

BloquearEsquina(10,10)

Pos (10,10)

mientras (HayFlorEnLaEsquina)

tomarFlor

Pos (avIni,caIni)

LiberarEsquina(10,10)

BloquearEsquina(11,11)

Pos (11,11)

depositarFlor

Pos (avIni,caIni)

LiberarEsquina(11,11)

BloquearEsquina(10,10)

Pos (10,10)

Pos (avIni,caIni)

LiberarEsquina(10,10)

fin

proceso juntarFloresEsquina

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

fin

proceso recolectarFlores

comenzar

Pos (11,11)

juntarFloresEsquina

Pos (1,1)

fin

areas

areaPC1: AreaPC(10,10,10,10)

areaPC2: AreaPC(11,11,11,11)

area1: AreaP(9,9,9,9)

area2: AreaP(9,10,9,10)

area3: AreaP(9,11,9,11)

area4: AreaP(9,12,9,12)

areaC: AreaP(1,1,1,1)

robots

robot recolector

variables

rob: numero

comenzar

RecibirMensaje(rob,robotC)

trasladarFlor

EnviarMensaje(rob,robotC)

fin

robot coordinador

variables

rob: numero

comenzar

EnviarMensaje(1,robot1)

EnviarMensaje(2,robot2)

EnviarMensaje(3,robot3)

EnviarMensaje(4,robot4)

repetir 4

RecibirMensaje(rob,*)

Informar ('RobotFinalizóÚltimo',rob)

recolectarFlores

fin

variables

robot1, robot2, robot3, robot4: recolector

robotC: coordinador

comenzar

AsignarArea(robot1,areaPC1)

AsignarArea(robot1,areaPC2)

AsignarArea(robot1,area1)

AsignarArea(robot2,areaPC1)

AsignarArea(robot2,areaPC2)

AsignarArea(robot2,area2)

AsignarArea(robot3,areaPC1)

AsignarArea(robot3,areaPC2)

AsignarArea(robot3,area3)

AsignarArea(robot4,areaPC1)

AsignarArea(robot4,areaPC2)

AsignarArea(robot4,area4)

AsignarArea(robotC,areaPC2)

AsignarArea(robotC,areaC)

Iniciar(robot1,9,9)

Iniciar(robot2,9,10)

Iniciar(robot3,9,11)

Iniciar(robot4,9,12)

Iniciar(robotC,1,1)

fin

(c) *Analizar: ¿Cuál de las 2 soluciones maximiza la concurrencia? ¿Se podría resolver este problema sin que los robots deban regresar a su esquina inicial?*

La solución del inciso (b) maximiza la concurrencia. No se puede resolver este problema sin que los robots deban regresar a su esquina inicial.

Ejercicio 5.

Realizar un programa en el que 4 robots juegan una carrera por avenidas diferentes: 4, 6, 8 y 10, respectivamente. Todos los robots inician en la calle 1. Para poder avanzar, cada robot debe juntar un papel de una fuente de papeles localizada en la esquina (11,11), colocarlo en la esquina actual de su avenida y avanzar un paso. Cuando la esquina fuente ya no tiene más papeles o cuando se haya completado la avenida, deberán avisar al robot coordinador y éste determinará el robot que llegó más lejos.

programa TP13_E5**procesos**

proceso juntarPapelyAvanzar (E av: numero; ES ca: numero)

comenzar

BloquearEsquina(11,11)

Pos (11,11)

mientras (HayPapelEnLaEsquina & (ca<100))

tomarPapel

Pos (av,ca)

LiberarEsquina(11,11)

depositarPapel

mover

ca:=PosCa

BloquearEsquina(11,11)

Pos (11,11)

Pos (av,ca)

LiberarEsquina(11,11)

fin

proceso actualizarMaximo (E ca: numero; E rob: numero; ES caMax: numero; ES robMax: numero)

comenzar

si (ca>caMax)

caMax:=ca

robMax:=rob

fin**areas**

areaPC: AreaPC(11,11,11,11)

area1: AreaP(4,1,4,100)

area2: AreaP(6,1,6,100)

area3: AreaP(8,1,8,100)

area4: AreaP(10,1,10,100)

areaC: AreaP(1,1,1,1)

robots

robot jugador

variables

rob, av, ca: numero

comenzar

av:=PosAv

ca:=PosCa

RecibirMensaje(rob,robotC)

```
juntarPapelyAvanzar(av,ca)
EnviarMensaje(rob,robotC)
EnviarMensaje(ca,robotC)
fin
robot coordinador
variables
  rob, robMax, ca, caMax: numero
comenzar
  caMax:=0
  robMax:=0
  EnviarMensaje(1,robot1)
  EnviarMensaje(2,robot2)
  EnviarMensaje(3,robot3)
  EnviarMensaje(4,robot4)
  repetir 4
    RecibirMensaje(rob,*)
    si (rob=1)
      RecibirMensaje(ca,robot1)
    sino
      si(rob=2)
        RecibirMensaje(ca,robot2)
      sino
        si (rob=3)
          RecibirMensaje(ca,robot3)
        sino
          RecibirMensaje(ca,robot4)
    actualizarMaximo(ca,rob,caMax,robMax)
  Informar ('RobotQueLlegóMásLejos',robMax)
fin
variables
  robot1, robot2, robot3, robot4: jugador
  robotC: coordinador
comenzar
  AsignarArea(robot1,areaPC)
  AsignarArea(robot1,area1)
  AsignarArea(robot2,areaPC)
  AsignarArea(robot2,area2)
  AsignarArea(robot3,areaPC)
  AsignarArea(robot3,area3)
  AsignarArea(robot4,areaPC)
  AsignarArea(robot4,area4)
  AsignarArea(robotC,areaC)
  Iniciar(robot1,4,1)
  Iniciar(robot2,6,1)
  Iniciar(robot3,8,1)
  Iniciar(robot4,10,1)
  Iniciar(robotC,1,1)
fin
```

Ejercicio 6.

(a) Tres robots deben recorrer el perímetro de su cuadrante, como se indica a continuación:

- El robot 1 comienza la esquina (2,2) y debe realizar un cuadrante de 6x6 juntando todas las flores que encuentre.
- El robot 2 comienza en la esquina (5,5) y debe realizar un cuadrante de 10x10 juntando todas las flores y los papeles que encuentre.
- El robot 3 comienza en la esquina (9,9) y debe realizar un cuadrante de 7x7 juntando todos los papeles que encuentre.

Cada robot que finalice su cuadrante deberá avisar al robot fiscalizador. Al recibir el aviso, el robot fiscalizador indicará, inmediatamente, una calle a la que deberá dirigirse el robot recolector, considerando que el robot que finalizó primero irá a la calle 20, el segundo a la 21 y el tercero a la 22.

Cuando los robots recolectores reciben un número de calle, deberán posicionarse en la avenida 1 de dicha calle y avanzar a lo largo de la calle depositando, en cada esquina, un papel, una flor o ambos, según lo que cada robot haya juntado. El recorrido finalizará al completar la calle o vaciarse las bolsas.

programa TP13_E6a**procesos**

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

proceso juntarFloresyPapelesEsquina (ES flores: numero; ES papeles: numero)

comenzar

juntarFloresEsquina(flores)

juntarPapelesEsquina(papeles)

fin

proceso recorrerCalleyDepositarFlores (E flores: numero)

comenzar

mientras ((flores>0) & (PosAv<100))

depositarFlor

flores:=flores-1

mover

si (flores>0)

depositarFlor

fin

proceso recorrerCalleyDepositarPapeles (E papeles: numero)

comenzar

mientras ((papeles>0) & (PosAv<100))

depositarPapel

papeles:=papeles-1

mover

si (papeles>0)

depositarPapeles

fin

proceso recorrerCalleyDepositarFloresyPapeles (E flores: numero; E papeles: numero)

comenzar

mientras (((flores>0) & (papeles>0)) & (PosAv<100))

depositarFlor

depositarPapel

flores:=flores-1

papeles:=papeles-1

mover

si ((flores>0) & (papeles>0))

depositarFlor

depositarPapel

fin

areas

areaPC1: AreaPC(2,2,16,16)

areaPC2: AreaPC(1,20,100,22)

areaF: AreaP(1,1,1,1)

robots

robot recolector1

variables

rob, flores, ca: numero

comenzar

flores:=0

RecibirMensaje(rob,robotF)

repetir 6

juntarFloresEsquina(flores)

mover

derecha

repetir 2

juntarFloresEsquina (flores)

mover

juntarFloresEsquina(flores)

BloquearEsquina(5,8)

mover

juntarFloresEsquina(flores)

mover

LiberarEsquina(5,8)

repetir 2

juntarFloresEsquina(flores)

mover

derecha


```
repetir 2
  juntarFloresEsquina(flores)
  mover
juntarFloresEsquina(flores)
BloquearEsquina(8,5)
mover
juntarFloresEsquina(flores)
mover
LiberarEsquina(8,5)
repetir 2
  juntarFloresEsquina(flores)
  mover
derecha
repetir 6
  juntarFloresEsquina(flores)
  mover
EnviarMensaje(rob,robotF)
repetir 2
  derecha
RecibirMensaje(ca,robotF)
Pos (1,ca)
recorrerCalleyDepositarFlores(flores)
fin
robot recolector2
variables
  rob, flores, papeles, ca: numero
comenzar
  flores:=0
  papeles:=0
  RecibirMensaje(rob,robotF)
  repetir 2
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
  juntarFloresyPapelesEsquina(flores,papeles)
  BloquearEsquina(5,8)
  mover
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
  LiberarEsquina(5,8)
  repetir 6
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
  derecha
  repetir 3
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
  juntarFloresyPapelesEsquina(flores,papeles)
  BloquearEsquina(9,15)
  mover
  juntarFloresyPapelesEsquina(flores,papeles)
```

```
mover
LiberarEsquina(9,15)
repetir 5
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
  derecha
  repetir 5
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
    juntarFloresyPapelesEsquina(flores,papeles)
    BloquearEsquina(15,9)
  mover
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
  LiberarEsquina(15,9)
  repetir 3
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
    derecha
    repetir 6
      juntarFloresyPapelesEsquina(flores,papeles)
      mover
      juntarFloresyPapelesEsquina(flores,papeles)
      BloquearEsquina(8,5)
    mover
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
    LiberarEsquina(8,5)
  repetir 2
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
  EnviarMensaje(rob,robotF)
  repetir 2
    derecha
  RecibirMensaje(ca,robotF)
  Pos (1,ca)
  recorrerCalleyDepositarFloresyPapeles(flores,papeles)
fin
robot recolector3
variables
  rob, papeles, ca: numero
comenzar
  papeles:=0
  RecibirMensaje(rob,robotF)
  repetir 5
    juntarPapelesEsquina(papeles)
    mover
    juntarPapelesEsquina(papeles)
    BloquearEsquina(9,15)
  mover
```

```

juntarPapelesEsquina(papeles)
mover
LiberarEsquina(9,15)
derecha
repetir 7
  juntarPapelesEsquina(papeles)
  mover
derecha
repetir 7
  juntarPapelesEsquina(papeles)
  mover
derecha
juntarPapelesEsquina(papeles)
BloquearEsquina(15,9)
mover
juntarPapelesEsquina(papeles)
mover
LiberarEsquina(15,9)
repetir 5
  juntarPapelesEsquina(papeles)
  mover
EnviarMensaje(rob,robotF)
repetir 2
  derecha
RecibirMensaje(ca,robotF)
Pos (1,ca)
recorrerCalleyDepositarPapeles(papeles)

```

fin

robot fiscalizador

variables

rob, ca: numero

comenzar

ca:=20

EnviarMensaje(1,robot1)

EnviarMensaje(2,robot2)

EnviarMensaje(3,robot3)

repetir 3

RecibirMensaje(rob,*)

si (rob=1)

EnviarMensaje(ca,robot1)

sino

si (rob=2)

EnviarMensaje(ca,robot2)

sino

EnviarMensaje(ca,robot3)

ca:=ca+1

fin

variables

robot1: recolector1

robot2: recolector2

```
robot3: recolector3
robotF: fiscalizador
```

comenzar

```
AsignarArea(robot1,areaPC1)
AsignarArea(robot1,areaPC2)
AsignarArea(robot2,areaPC1)
AsignarArea(robot2,areaPC2)
AsignarArea(robot3,areaPC1)
AsignarArea(robot3,areaPC2)
AsignarArea(robotF,areaF)
Iniciar(robot1,2,2)
Iniciar(robot2,5,5)
Iniciar(robot3,9,9)
Iniciar(robotF,1,1)
```

fin

(b) *Analizar (no es necesario implementar): ¿Cómo debería modificarse el ejercicio anterior si los robots recolectores no conocen, de antemano, el tamaño de su cuadrante (por ejemplo, porque lo calcula el fiscalizador de manera aleatoria)?*

Si los robots recolectores no conocen, de antemano, el tamaño de su cuadrante (por ejemplo, porque lo calcula el fiscalizador de manera aleatoria), el ejercicio anterior debería modificarse de manera tal de ir bloqueando y liberando todas las esquinas que van transitando los robots, debido a que no es posible conocer, de antemano, las esquinas de posible colisión.

(c) *Modificar el ejercicio anterior (a) para que, ahora, el robot fiscalizador espere a que todos los robots recolectores hayan completado sus cuadrantes antes de indicarles la calle que deberán recorrer.*

programa TP13_E6c

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

```
mientras (HayFlorEnLaEsquina)
  tomarFlor
  flores:=flores+1
```

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

```
mientras (HayPapelEnLaEsquina)
  tomarPapel
  papeles:=papeles+1
```

fin

proceso juntarFloresyPapelesEsquina (ES flores: numero; ES papeles: numero)

comenzar

```

juntarFloresEsquina(flores)
juntarPapelesEsquina(papeles)
fin
proceso recorrerCalleyDepositarFlores (E flores: numero)
comenzar
  mientras ((flores>0) & (PosAv<100))
    depositarFlor
    flores:=flores-1
    mover
  si (flores>0)
    depositarFlor
fin
proceso recorrerCalleyDepositarPapeles (E papeles: numero)
comenzar
  mientras ((papeles>0) & (PosAv<100))
    depositarPapel
    papeles:=papeles-1
    mover
  si (papeles>0)
    depositarPapeles
fin
proceso recorrerCalleyDepositarFloresyPapeles (E flores: numero; E papeles: numero)
comenzar
  mientras (((flores>0) & (papeles>0)) & (PosAv<100))
    depositarFlor
    depositarPapel
    flores:=flores-1
    papeles:=papeles-1
    mover
  si ((flores>0) & (papeles>0))
    depositarFlor
    depositarPapel
fin
areas
areaPC1: AreaPC(2,2,16,16)
areaPC2: AreaPC(1,20,100,22)
areaF: AreaP(1,1,1,1)
robots
robot recolector1
variables
  rob, flores, ca: numero
comenzar
  flores:=0
  RecibirMensaje(rob,robotF)
  repetir 6
    juntarFloresEsquina(flores)
    mover
  derecha
  repetir 2
    juntarFloresEsquina(flores)

```

```
mover
juntarFloresEsquina(flores)
BloquearEsquina(5,8)
mover
juntarFloresEsquina(flores)
mover
LiberarEsquina(5,8)
repetir 2
  juntarFloresEsquina(flores)
  mover
derecha
repetir 2
  juntarFloresEsquina(flores)
  mover
juntarFloresEsquina(flores)
BloquearEsquina(8,5)
mover
juntarFloresEsquina(flores)
mover
LiberarEsquina(8,5)
repetir 2
  juntarFloresEsquina(flores)
  mover
derecha
repetir 6
  juntarFloresEsquina(flores)
  mover
EnviarMensaje(rob,robotF)
repetir 2
  derecha
RecibirMensaje(ca,robotF)
Pos (1,ca)
recorrerCalleyDepositarFlores(flores)
fin
robot recolector2
variables
  rob, flores, papeles, ca: numero
comenzar
  flores:=0
  papeles:=0
  RecibirMensaje(rob,robotF)
  repetir 2
    juntarFloresyPapelesEsquina(flores,papeles)
    mover
  juntarFloresyPapelesEsquina(flores,papeles)
  BloquearEsquina(5,8)
  mover
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
  LiberarEsquina(5,8)
```

```
repetir 6
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
derecha
repetir 3
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
juntarFloresyPapelesEsquina(flores,papeles)
BloquearEsquina(9,15)
mover
juntarFloresyPapelesEsquina(flores,papeles)
mover
LiberarEsquina(9,15)
repetir 5
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
derecha
repetir 5
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
juntarFloresyPapelesEsquina(flores,papeles)
BloquearEsquina(15,9)
mover
juntarFloresyPapelesEsquina(flores,papeles)
mover
LiberarEsquina(15,9)
repetir 3
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
derecha
repetir 6
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
juntarFloresyPapelesEsquina(flores,papeles)
BloquearEsquina(8,5)
mover
juntarFloresyPapelesEsquina(flores,papeles)
mover
LiberarEsquina(8,5)
repetir 2
  juntarFloresyPapelesEsquina(flores,papeles)
  mover
EnviarMensaje(rob,robotF)
repetir 2
  derecha
RecibirMensaje(ca,robotF)
Pos (1,ca)
recorrerCalleyDepositarFloresyPapeles(flores,papeles)
fin
robot recolector3
```

variables

rob, papeles, ca: numero

comenzar

papeles:=0

RecibirMensaje(rob,robotF)

repetir 5

 juntarPapelesEsquina(papeles)

 mover

 juntarPapelesEsquina(papeles)

 BloquearEsquina(9,15)

 mover

 juntarPapelesEsquina(papeles)

 mover

 LiberarEsquina(9,15)

 derecha

 repetir 7

 juntarPapelesEsquina(papeles)

 mover

 derecha

 repetir 7

 juntarPapelesEsquina(papeles)

 mover

 derecha

 juntarPapelesEsquina(papeles)

 BloquearEsquina(15,9)

 mover

 juntarPapelesEsquina(papeles)

 mover

 LiberarEsquina(15,9)

 repetir 5

 juntarPapelesEsquina(papeles)

 mover

 EnviarMensaje(rob,robotF)

 repetir 2

 derecha

 RecibirMensaje(ca,robotF)

 Pos (1,ca)

 recorrerCalleyDepositarPapeles(papeles)

fin

robot fiscalizador

variables

rob, ca, ca1, ca2, ca3: numero

comenzar

ca:=20

EnviarMensaje(1,robot1)

EnviarMensaje(2,robot2)

EnviarMensaje(3,robot3)

repetir 3

 RecibirMensaje(rob,*)

 si (rob=1)


```
    ca1:=ca
sino
    si (rob=2)
        ca2:=ca
    sino
        ca3:=ca
    ca:=ca+1
    EnviarMensaje(ca1,robot1)
    EnviarMensaje(ca2,robot2)
    EnviarMensaje(ca3,robot3)
fin
variables
robot1: recolector1
robot2: recolector2
robot3: recolector3
robotF: fiscalizador
comenzar
    AsignarArea(robot1,areaPC1)
    AsignarArea(robot1,areaPC2)
    AsignarArea(robot2,areaPC1)
    AsignarArea(robot2,areaPC2)
    AsignarArea(robot3,areaPC1)
    AsignarArea(robot3,areaPC2)
    AsignarArea(robotF,areaF)
    Iniciar(robot1,2,2)
    Iniciar(robot2,5,5)
    Iniciar(robot3,9,9)
    Iniciar(robotF,1,1)
fin
```

Trabajo Práctico N° 14:

Módulo Concurrente (Realizar programas en R-info con distintos tipos de robots. Utilizar memoria compartida para la comunicación y sincronización entre robots. Combinar problemas con memoria compartida y pasaje de mensajes. Distinguir modelos de algoritmos a desarrollar de acuerdo al problema planteado).

Ejercicio 1: Clientes y Servidores.

Existe un robot que sirve de flores a tres robots clientes. Cada cliente solicita al servidor que le deposite en su esquina siguiente una cantidad de flores aleatoria (entre 1 y 4). Por ejemplo, si el cliente se encuentra en la esquina (2,1), le solicitará que coloque x cantidad de flores en la esquina (2,2). Cuando el robot servidor deposita las flores en la esquina solicitada, el cliente las junta y las deposita una a una a lo largo de la avenida en la que se encuentra. El programa finaliza cuando todos los robots clientes completan su avenida. Asumir que el robot servidor tiene flores suficientes en su bolsa. El robot servidor se inicia en la esquina (100,100). Los robots clientes inician en las esquinas (1,1), (2,1) y (3,1), respectivamente.

programa TP14_E1

procesos

proceso juntarFloresEsquina

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

fin

proceso depositarFloryAvanzar (E flores: numero)

comenzar

mientras ((flores>0) & (PosCa<100))

depositarFlor

flores:=flores-1

mover

fin

areas

area1: AreaPC(1,1,1,100)

area2: AreaPC(2,1,2,100)

area3: AreaPC(3,1,3,100)

areaS: AreaP(100,100,100,100)

robots

robot cliente

variables

rob, av, ca, flores: numero

ok: boolean

comenzar

RecibirMensaje(rob,robotS)

mientras (PosCa<100)

```

Random(flores,1,4)
av:=PosAv
ca:=PosCa+1
EnviarMensaje(rob,robotS)
EnviarMensaje(flores,robotS)
EnviarMensaje(av,robotS)
EnviarMensaje(ca,robotS)
RecibirMensaje(ok,robotS)
Pos (av,ca)
juntarFloresEsquina
Pos (av,ca-1)
depositarFloryAvanzar(flores)
EnviarMensaje(rob,robotS)
EnviarMensaje(0,robotS)
fin
robot servidor
variables
rob, avIni, caIni, av, ca, flores, term: numero
ok: boolean
comenzar
avIni:=PosAv
caIni:=PosCa
term:=0
ok:=V
EnviarMensaje(1,robot1)
EnviarMensaje(2,robot2)
EnviarMensaje(3,robot3)
mientras (term<3)
  RecibirMensaje(rob,*)
  si (rob=1)
    RecibirMensaje(flores,robot1)
    si (flores<>0)
      RecibirMensaje(av,robot1)
      RecibirMensaje(ca,robot1)
      Pos (av,ca)
      repetir flores
        depositarFlor
      Pos (avIni,caIni)
      EnviarMensaje(ok,robot1)
    sino
      term:=term+1
  sino
    si (rob=2)
      RecibirMensaje(flores,robot2)
      si (flores<>0)
        RecibirMensaje(av,robot2)
        RecibirMensaje(ca,robot2)
        Pos (av,ca)
        repetir flores
          depositarFlor

```

```
    Pos (avIni,caIni)
    EnviarMensaje(ok,robot2)
sino
    term:=term+1
sino
    RecibirMensaje(flores,robot3)
    si (flores<>0)
        RecibirMensaje(av,robot3)
        RecibirMensaje(ca,robot3)
        Pos (av,ca)
        repetir flores
            depositarFlor
        Pos (avIni,caIni)
        EnviarMensaje(ok,robot3)
    sino
        term:=term+1
fin
variables
robot1, robot2, robot3: cliente
robotS: servidor
comenzar
AsignarArea(robot1,area1)
AsignarArea(robot2,area2)
AsignarArea(robot3,area3)
AsignarArea(robotS,areaS)
AsignarArea(robotS,area1)
AsignarArea(robotS,area2)
AsignarArea(robotS,area3)
Iniciar(robot1,1,1)
Iniciar(robot2,2,1)
Iniciar(robot3,3,1)
Iniciar(robotS,100,100)
fin
```

Ejercicio 2: Productores y Consumidores.

Existen dos robots productores que recorren las avenidas 5 y 10, respectivamente, juntando todos los papeles de su avenida. A lo largo del recorrido, cada vez que juntan 5 papeles, los depositan en la esquina (50,50). Además, existen dos robots consumidores que intentan tomar una cantidad aleatoria de papeles (entre 2 y 5) de la esquina (50,50) para depositarla en su esquina de origen. Si la esquina (50,50) no posee la cantidad de papeles requerida, vuelven a su esquina de origen sin tomar ningún papel. Si luego de 8 intentos seguidos, detectan que la esquina (50,50) no tiene papeles suficientes para juntar, entonces, asumirán que los productores ya han completado su trabajo y, por lo tanto, terminarán su tarea también. Los consumidores inician en las esquinas (11,1) y (12,1), respectivamente.

programa TP14_E2**areas**

areaC: AreaC(50,50,50,50)

areaP1: AreaP(5,1,5,100)

areaP2: AreaP(10,1,10,100)

areaC1: AreaP(11,1,11,1)

areaC2: AreaP(12,1,12,1)

robots

robot productor

variables

av, ca, papeles: numero

comenzar

repetir 99

av:=PosAv

ca:=PosCa

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

si (papeles=5)

papeles:=0

BloquearEsquina(50,50)

Pos (50,50)

repetir papeles

depositarPapel

Pos (av,ca)

LiberarEsquina(50,50)

mover

fin

robot consumidor

variables

avIni, caIni, papeles, papelesReq, int: numero

comenzar

avIni:=PosAv

caIni:=PosCa

int:=0

mientras (int<8)

```
papeles:=0
Random(papelesReq,2,5)
BloquearEsquina(50,50)
Pos (50,50)
mientras (HayPapelEnLaEsquina & (papeles<papelesReq))
  tomarPapel
  papeles:=papeles+1
si (papeles<papelesReq)
  repetir papeles
    depositarPapel
  Pos (avIni,caIni)
  LiberarEsquina(50,50)
  int:=int+1
sino
  Pos (avIni,caIni)
  LiberarEsquina(50,50)
  repetir papeles
    depositarPapel
  int:=0
fin
variables
robotP1, robotP2: productor
robotC1, robotC2: consumidor
comenzar
AsignarArea(robotP1,areaC)
AsignarArea(robotP1,areaP1)
AsignarArea(robotP2,areaC)
AsignarArea(robotP2,areaP2)
AsignarArea(robotC1,areaC)
AsignarArea(robotC1,areaC1)
AsignarArea(robotC2,areaC)
AsignarArea(robotC2,areaC2)
Iniciar(robotP1,5,1)
Iniciar(robotP2,10,1)
Iniciar(robotC1,11,1)
Iniciar(robotC2,12,1)
fin
```

Ejercicio 3: Sincronización Barrera.

Tres robots deben vaciar de papeles su avenida, comenzando por la calle 1 y terminando en la calle 100. El trabajo lo deben realizar todos juntos y en etapas: los robots inician juntos y, cuando todos completan una etapa del trabajo, pueden avanzar a la siguiente, lo que significa que, para poder pasar de etapa, los robots deben esperar que todos hayan completado la etapa en curso. Se proponen dos posibles soluciones a este problema (etapas homogéneas o etapas heterogéneas):

En cada solución, analizar cómo debería finalizar el programa. Los robots inician en las esquinas (1,1), (2,1) y (3,1), respectivamente. Existe un robot coordinador, cuya única tarea es asignar identificadores a cada robot.

(a) Implementar el programa considerando que cada robot completa una etapa cada 5 esquinas.

programa TP14_E3a

procesos

proceso juntarPapelesEsquina

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

fin

areas

area1: AreaP(1,1,1,100)

area2: AreaP(2,1,2,100)

area3: AreaP(3,1,3,100)

areaC: AreaP(5,5,5,5)

robots

robot sincronizado

variables

rob: numero

ok: boolean

comenzar

ok:=V

RecibirMensaje(rob,robotC)

repetir 19

repetir 5

juntarPapelesEsquina

mover

si (rob=1)

EnviarMensaje(ok,robot2)

EnviarMensaje(ok,robot3)

sino

si (rob=2)

EnviarMensaje(ok,robot1)

EnviarMensaje(ok,robot3)

sino

EnviarMensaje(ok,robot1)

```

    EnviarMensaje(ok,robot2)
  repetir 2
    RecibirMensaje(ok,*)
  repetir 4
    juntarPapelesEsquina
    mover
    juntarPapelesEsquina
  si (rob=1)
    EnviarMensaje(ok,robot2)
    EnviarMensaje(ok,robot3)
  sino
    si (rob=2)
      EnviarMensaje(ok,robot1)
      EnviarMensaje(ok,robot3)
    sino
      EnviarMensaje(ok,robot1)
      EnviarMensaje(ok,robot2)
  repetir 2
    RecibirMensaje(ok,*)
fin
robot coordinador
comenzar
  EnviarMensaje(1,robot1)
  EnviarMensaje(2,robot2)
  EnviarMensaje(3,robot3)
fin
variables
robot1, robot2, robot3: sincronizado
robotC: coordinador
comenzar
  AsignarArea(robot1,area1)
  AsignarArea(robot2,area2)
  AsignarArea(robot3,area3)
  AsignarArea(robotC,areaC)
  Iniciar(robot1,1,1)
  Iniciar(robot2,2,1)
  Iniciar(robot3,3,1)
  Iniciar(robotC,5,5)
fin

```

(b) Implementar el programa considerando que cada robot completa una etapa luego de juntar N papeles. El valor de N (entre 1 y 5) lo calcula cada robot antes de iniciar cada etapa.

programa TP14_E3b

procesos

proceso juntarPapelEsquina (ES papeles: numero)

comenzar


```

    si (HayPapelEnLaEsquina)
        tomarPapel
        papeles:=papeles+1
    sino
        mover
fin
areas
area1: AreaP(1,1,1,100)
area2: AreaP(2,1,2,100)
area3: AreaP(3,1,3,100)
areaC: AreaP(5,5,5,5)
robots
robot sincronizado
variables
    rob, papeles, papelesEtapa: numero
    ok, ok1, ok2: boolean
comenzar
    ok:=V
    ok1:=V
    ok2:=V
    RecibirMensaje(rob,robotC)
    mientras ((PosCa<100) & (ok1) & (ok2))
        papeles:=0
        Random(papelesEtapa,1,5)
        mientras ((papeles<papelesEtapa) & (PosCa<100))
            juntarPapelEsquina(papeles)
        si (papeles<papelesEtapa)
            ok:=F
        si (rob=1)
            EnviarMensaje(ok,robot2)
            EnviarMensaje(ok,robot3)
        sino
            si (rob=2)
                EnviarMensaje(ok,robot1)
                EnviarMensaje(ok,robot3)
            sino
                EnviarMensaje(ok,robot1)
                EnviarMensaje(ok,robot2)
        RecibirMensaje(ok1,*)
        RecibirMensaje(ok2,*)
fin
robot coordinador
comenzar
    EnviarMensaje(1,robot1)
    EnviarMensaje(2,robot2)
    EnviarMensaje(3,robot3)
fin
variables
    robot1, robot2, robot3: sincronizado
    robotC: coordinador

```

comenzar

AsignarArea(robot1,area1)

AsignarArea(robot2,area2)

AsignarArea(robot3,area3)

AsignarArea(robotC,areaC)

Iniciar(robot1,1,1)

Iniciar(robot2,2,1)

Iniciar(robot3,3,1)

Iniciar(robotC,5,5)

fin

Ejercicio 4: Jefe y Trabajadores - Master / Slave.

Un robot jefe asigna tareas a los trabajadores. Las tareas consisten en (1) recoger flores, (2) recoger papeles, (3) vaciar bolsa, (4) finalizar. Existen 2 robots trabajadores que reciben solicitudes de tareas del robot jefe. Para cada solicitud, reciben la tarea y la esquina donde deben realizarla (salvo cuando la tarea es 4, que no deben acceder a una esquina). Luego de recibir la tarea, los robots van a la esquina indicada, realizan la tarea, avisan al jefe que ya la completaron y quedan a la espera de una nueva tarea. El robot jefe inicia en la esquina (1,1) y los robots trabajadores inician en las esquinas (2,1) y (3,1), respectivamente. Las tareas se asignan aleatoriamente a cualquier esquina dentro del cuadrante comprendido entre las esquinas (2,2) y (100,100). El robot jefe envía 10 tareas aleatorias a trabajadores aleatorios y termina. Analizar: ¿Existe el riesgo de que el programa quede bloqueado y que ningún robot trabajador pueda realizar su tarea? ¿En qué caso puede suceder esto? ¿Qué resulta necesario considerar para evitar esta situación?

programa TP14_E4**procesos**

proceso recogerFlores

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

fin

proceso recogerPapeles

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

fin

proceso vaciarBolsa

comenzar

mientras (HayFlorEnLaBolsa)

depositarFlor

mientras (HayPapelEnLaBolsa)

depositarPapel

fin

areas

areaPC: AreaPC(2,2,100,100)

area1: AreaP(2,1,2,1)

area2: AreaP(3,1,3,1)

areaM: AreaP(1,1,1,1)

robots

robot slave

variables

avIni, caIni, av, ca, tarea: numero

comenzar

avIni:=PosAv

caIni:=PosCa

RecibirMensaje(tarea,robotM)

mientras (tarea<>4)

```
RecibirMensaje(av,robotM)
RecibirMensaje(ca,robotM)
BloquearEsquina(av,ca)
Pos (av,ca)
si (tarea=1)
  recogerFlores
sino
  si (tarea=2)
    recogerPapeles
  sino
    vaciarBolsa
Pos (avIni,caIni)
LiberarEsquina(av,ca)
RecibirMensaje(tarea,robotM)
fin
robot master
variables
  rob, av, ca, tarea: numero
  rob1Act, rob2Act: boolean
comenzar
  rob1Act:=V
  rob2Act:=V
  repetir 10
    rob:=0
    si ((rob1Act) & (rob2Act))
      Random(rob,1,2)
    sino
      si (rob1Act)
        rob:=1
      sino
        si (rob2Act)
          rob:=2
    si (rob<>0)
      Random(tarea,1,4)
      Random(av,2,100)
      Random(ca,2,100)
      si (rob=1)
        EnviarMensaje(tarea,robot1)
        si (tarea<>4)
          EnviarMensaje(av,robot1)
          EnviarMensaje(ca,robot1)
        sino
          rob1Act:=F
      sino
        EnviarMensaje(tarea,robot2)
        si (tarea<>4)
          EnviarMensaje(av,robot2)
          EnviarMensaje(ca,robot2)
        sino
          rob2Act:=F
```

```
    si (rob1Act)
        EnviarMensaje(4,robot1)
    si (rob2Act)
        EnviarMensaje(4,robot2)
fin
variables
robot1, robot2: slave
robotM: master
comenzar
    AsignarArea(robot1,areaPC)
    AsignarArea(robot1,area1)
    AsignarArea(robot2,areaPC)
    AsignarArea(robot2,area2)
    AsignarArea(robotM,areaM)
    Iniciar(robot1,2,1)
    Iniciar(robot2,3,1)
    Iniciar(robotM,1,1)
fin
```

Trabajo Práctico N° 15: **Módulo Concurrente (Repaso).**

Ejercicio 1.

Se organizó una competencia entre el equipo rojo y el equipo azul. Cada equipo consta de dos robots y debe realizar una tarea:

- *Los robots R1 y R2 del equipo rojo debe juntar todas las flores de las avenidas 2 y 3, respectivamente.*
- *Los robots A1 y A2 del equipo azul debe juntar todos los papeles de las calles 98 y 99, respectivamente.*

Al finalizar la competencia, un robot fiscalizador deberá informar el equipo que juntó más objetos.

programa TP15_E1

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina (ES papeles: numero)

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

papeles:=papeles+1

fin

proceso recorrerAvenida (ES flores: numero)

comenzar

repetir 96

juntarFloresEsquina(flores)

mover

juntarFloresEsquina(flores)

BloquearEsquina(PosAv,98)

mover

juntarFloresEsquina(flores)

BloquearEsquina(PosAv,99)

mover

LiberarEsquina(PosAv,98)

juntarFloresEsquina(flores)

mover

LiberarEsquina(PosAv,99)

juntarFloresEsquina(flores)

fin

proceso recorrerCalle (ES papeles: numero)

comenzar

```
juntarPapelesEsquina(papeles)
BloquearEsquina(2,PosCa)
mover
juntarPapelesEsquina(papeles)
BloquearEsquina(3,PosCa)
mover
LiberarEsquina(2,PosCa)
juntarPapelesEsquina(papeles)
mover
LiberarEsquina(3,PosCa)
repetir 96
  juntarPapelesEsquina(papeles)
  mover
juntarPapelesEsquina(papeles)
```

fin

areas

```
areaPC: AreaPC(2,98,3,99)
areaR1a: AreaP(2,1,2,97)
areaR1b: AreaP(2,100,2,100)
areaR2a: AreaP(3,1,3,97)
areaR2b: AreaP(3,100,3,100)
areaA1a: AreaP(1,98,1,98)
areaA1b: AreaP(4,98,100,98)
areaA2a: AreaP(1,99,1,99)
areaA2b: AreaP(4,99,100,99)
areaF: AreaP(100,100,100,100)
```

robots

robot equipoR

variables

rob, flores: numero

comenzar

```
flores:=0
RecibirMensaje(rob,robotF)
recorrerAvenida(flores)
EnviarMensaje(rob,robotF)
EnviarMensaje(flores,robotF)
```

fin

robot equipoA

variables

rob, papeles: numero

comenzar

```
papeles:=0
derecha
RecibirMensaje(rob,robotF)
recorrerCalle(papeles)
EnviarMensaje(rob,robotF)
EnviarMensaje(papeles,robotF)
```

fin

robot fiscalizador

variables

rob, flores, papeles, elem: numero

comenzar

flores:=0

papeles:=0

EnviarMensaje(1,robotR1)

EnviarMensaje(2,robotR2)

EnviarMensaje(3,robotA1)

EnviarMensaje(4,robotA2)

repetir 4

RecibirMensaje(rob,*)

si (rob=1)

RecibirMensaje(elem,robotR1)

flores:=flores+elem

sino

si (rob=2)

RecibirMensaje(elem,robotR2)

flores:=flores+elem

sino

si (rob=3)

RecibirMensaje(elem,robotA1)

papeles:=papeles+elem

sino

RecibirMensaje(elem,robotA2)

papeles:=papeles+elem

si (flores>papeles)

Informar ('EquipoGanador',1)

sino

si (papeles>flores)

Informar ('EquipoGanador',2)

sino

Informar ('LosEquiposEmpataron',flores,papeles)

fin**variables**

robotR1, robotR2: equipoR

robotA1, robotA2: equipoA

robotF: fiscalizador

comenzar

AsignarArea(robotR1,areaPC)

AsignarArea(robotR1,areaR1a)

AsignarArea(robotR1,areaR1b)

AsignarArea(robotR2,areaPC)

AsignarArea(robotR2,areaR2a)

AsignarArea(robotR2,areaR2b)

AsignarArea(robotA1,areaPC)

AsignarArea(robotA1,areaA1a)

AsignarArea(robotA1,areaA1b)

AsignarArea(robotA2,areaPC)

AsignarArea(robotA2,areaA2a)

AsignarArea(robotA2,areaA2b)

AsignarArea(robotF,areaF)

Iniciar(robotR1,2,1)

Iniciar(robotR2,3,1)

Iniciar(robotA1,1,98)

Iniciar(robotA2,1,99)

Iniciar(robotF,100,100)

fin

Ejercicio 2.

Tres robots recolectores deben avanzar por su calle vaciando las esquinas. El avance debe realizarse en conjunto en etapas, siguiendo el modelo de sincronización barrera, en el cual los robots deben esperar que todos terminen su tarea antes de avanzar a la siguiente etapa. Cada etapa consiste en recorrer 10 esquinas y, luego, depositar todas las flores recolectadas en la esquina (50,50). Una vez que los robots recolectores completaron toda su calle, un robot fiscalizador deberá juntar todas las flores de la esquina (50,50) e informar la cantidad total de flores juntadas. Los robots recolectores inician en las esquinas (1,1), (1,2) y (1,3), respectivamente. El robot fiscalizador inicia en la esquina (1,4).

programa TP15_E2**procesos**

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

proceso juntarPapelesEsquina

comenzar

mientras (HayPapelEnLaEsquina)

tomarPapel

fin

proceso juntarFloresyPapelesEsquina (ES flores: numero)

comenzar

juntarFloresEsquina(flores)

juntarPapelesEsquina

fin

proceso recorrerCalle (E pasos: numero)

variables

av, ca, flores: numero

comenzar

flores:=0

repetir pasos

juntarFloresyPapelesEsquina(flores)

mover

av:=PosAv

ca:=PosCa

si (pasos=9)

juntarFloresyPapelesEsquina(flores)

BloquearEsquina(50,50)

Pos (50,50)

repetir flores

depositarFlor

Pos (av,ca)

LiberarEsquina(50,50)

fin

areas

```
areaC: AreaC(50,50,50,50)
area1: AreaP(1,1,100,1)
area2: AreaP(1,2,100,2)
area3: AreaP(1,3,100,3)
areaF: AreaP(1,4,1,4)
```

robots

```
robot sincronizado
```

variables

```
rob: numero
ok: boolean
```

comenzar

```
ok:=V
derecha
RecibirMensaje(rob,robotF)
repetir 9
  recorrerCalle(10)
  si (rob=1)
    EnviarMensaje(ok,robot2)
    EnviarMensaje(ok,robot3)
  sino
    si (rob=2)
      EnviarMensaje(ok,robot1)
      EnviarMensaje(ok,robot3)
    sino
      EnviarMensaje(ok,robot1)
      EnviarMensaje(ok,robot2)
  repetir 2
    RecibirMensaje(ok,*)
recorrerCalle(9)
si (rob=1)
  EnviarMensaje(ok,robot2)
  EnviarMensaje(ok,robot3)
sino
  si (rob=2)
    EnviarMensaje(ok,robot1)
    EnviarMensaje(ok,robot3)
  sino
    EnviarMensaje(ok,robot1)
    EnviarMensaje(ok,robot2)
repetir 2
  RecibirMensaje(ok,*)
  EnviarMensaje(ok,robotF)
```

fin

```
robot fiscalizador
```

variables

```
av, ca, flores: numero
ok: boolean
```

comenzar

```
flores:=0
```

```
EnviarMensaje(1,robot1)
EnviarMensaje(2,robot2)
EnviarMensaje(3,robot3)
repetir 3
  RecibirMensaje(ok,*)
  Pos (50,50)
  juntarFloresEsquina(flores)
  Informar ('FloresTotales',flores)
```

fin

variables

```
robot1, robot2, robot3: sincronizado
robotF: fiscalizador
```

comenzar

```
AsignarArea(robot1,areaC)
AsignarArea(robot1,area1)
AsignarArea(robot2,areaC)
AsignarArea(robot2,area2)
AsignarArea(robot3,areaC)
AsignarArea(robot3,area3)
AsignarArea(robotF,areaC)
AsignarArea(robotF,areaF)
Iniciar(robot1,1,1)
Iniciar(robot2,1,2)
Iniciar(robot3,1,3)
Iniciar(robotF,1,4)
```

fin

Ejercicio 3.

Dos robots recolectores avanzan por las calles 3 y 4, respectivamente, juntando todas las flores a su paso. Cada esquina tiene, por lo menos, una flor. Cada vez que juntan 10 flores o que avanzan 5 esquinas, deberán vaciar de flores su bolsa en el depósito localizado en la esquina (10,10). Cada vez que se depositan flores en el depósito, un robot cosechador deberá juntar dichas flores. Cuando ambos recolectores hayan completado sus calles, el robot cosechador deberá informar la cantidad de flores recolectadas. Los recolectores inician en la esquina (1,3) y (1,4), respectivamente, y el cosechador en la esquina (1,5).

programa TP15_E3

procesos

proceso vaciarFloresBolsa

variables

av, ca: numero

comenzar

av:=PosAv

ca:=PosCa

BloquearEsquina(10,10)

Pos (10,10)

mientras (HayFlorEnLaBolsa)

depositarFlor

Pos (av,ca)

LiberarEsquina(10,10)

fin

proceso recorrerCalle

variables

av, ca, flores, esquinas: numero

ok: boolean

comenzar

flores:=0

esquinas:=0

ok:=V

repetir 99

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

si (flores=10)

vaciarFloresBolsa

EnviarMensaje(ok,robotC)

flores:=0

esquinas:=esquinas+1

si (esquinas=5)

vaciarFloresBolsa

EnviarMensaje(ok,robotC)

esquinas:=0

mover

mientras (HayFlorEnLaEsquina)

```

    tomarFlor
    flores:=flores+1
    si (flores=10)
        vaciarFloresBolsa
        EnviarMensaje(ok,robotC)
    esquinas:=esquinas+1
    si (esquinas=5)
        vaciarFloresBolsa
        EnviarMensaje(ok,robotC)

```

fin

areas

```

    esquina: AreaC(10,10,10,10)
    area1: AreaP(1,3,100,3)
    area2: AreaP(1,4,100,4)
    areaC: AreaP(1,5,1,5)

```

robots

```

    robot recolector
    comenzar
    derecha
    EnviarMensaje(V,robotC)
    recorrerCalle
    EnviarMensaje(F,robotC)

```

fin

robot cosechador

variables

```

    av, ca, flores: numero
    ok1, ok2, cosechar: boolean

```

comenzar

```

    av:=PosAv
    ca:=PosCa
    flores:=0
    RecibirMensaje(ok1,*)
    RecibirMensaje(ok2,*)
    mientras ((ok1) | (ok2))
        RecibirMensaje(cosechar,*)
        si (cosechar)
            BloquearEsquina(10,10)
            Pos (10,10)
            mientras (HayFlorEnLaEsquina)
                tomarFlor
                flores:=flores+1
            Pos (av,ca)
            LiberarEsquina(10,10)
        sino
            si (ok1)
                ok1:=F
            sino
                ok2:=F
    Informar ('FloresRecolectadas',flores)

```

fin

variables

robot1, robot2: recolector
robotC: cosechador

comenzar

AsignarArea(robot1,esquina)
AsignarArea(robot1,area1)
AsignarArea(robot2,esquina)
AsignarArea(robot2,area2)
AsignarArea(robotC,esquina)
AsignarArea(robotC,areaC)
Iniciar(robot1,1,3)
Iniciar(robot2,1,4)
Iniciar(robotC,1,5)

fin

Ejercicio 4.

Tres robots floreros tienen 8 intentos en total para juntar todas las flores dentro del cuadrante comprendido entre las esquinas (40,40) y (60,60). Para ello, en cada intento, un robot fiscalizador indicará a un robot aleatorio la esquina a la que debe dirigirse. El fiscalizador calculará esta esquina de manera aleatoria. Al completarse los 8 intentos, los robots floreros deberán depositar todas las flores juntadas en la esquina (10,10) y el robot fiscalizador deberá informar la cantidad total de flores juntadas por los robots. Los robots floreros inician en las esquinas (1,1), (2,1) y (3,1), respectivamente, y el fiscalizador en la (4,1).

programa TP15_E4

procesos

proceso juntarFloresEsquina (ES flores: numero)

comenzar

mientras (HayFlorEnLaEsquina)

tomarFlor

flores:=flores+1

fin

areas

areaC: AreaC(10,10,10,10)

areaPC: AreaPC(40,40,60,60)

area1: AreaP(1,1,1,1)

area2: AreaP(2,1,2,1)

area3: AreaP(3,1,3,1)

areaF: AreaP(4,1,4,1)

robots

robot florero

variables

avIni, caIni, av, ca, flores: numero

ok: boolean

comenzar

avIni:=PosAv

caIni:=PosCa

flores:= 0

RecibirMensaje(ok,robotF)

mientras (ok)

RecibirMensaje(av,robotF)

RecibirMensaje(ca,robotF)

BloquearEsquina(av,ca)

Pos (av,ca)

juntarFloresEsquina(flores)

Pos (avIni,caIni)

LiberarEsquina(av,ca)

RecibirMensaje(ok,robotF)

BloquearEsquina(10,10)

Pos (10,10)

repetir flores

depositarFlor


```

    Pos (avIni,caIni)
    LiberarEsquina(10,10)
    EnviarMensaje(flores,robotF)
fin
robot fiscalizador
variables
    rob, av, ca, flores, floresTotal: numero
    ok: boolean
comenzar
    flores:=0
    ok:=V
    repetir 8
        Random(rob,1,3)
        Random(av,40,60)
        Random(ca,40,60)
        si (rob=1)
            EnviarMensaje(ok,robot1)
            EnviarMensaje(av,robot1)
            EnviarMensaje(ca,robot1)
        sino
            si (rob=2)
                EnviarMensaje(ok,robot2)
                EnviarMensaje(av,robot2)
                EnviarMensaje(ca,robot2)
            sino
                EnviarMensaje(ok,robot3)
                EnviarMensaje(av,robot3)
                EnviarMensaje(ca,robot3)
        ok:=F
        EnviarMensaje(ok,robot1)
        EnviarMensaje(ok,robot2)
        EnviarMensaje(ok,robot3)
    repetir 3
        RecibirMensaje(flores,*)
        floresTotal:=floresTotal+flores
    Informar ('FloresJuntadas',floresTotal)
fin
variables
    robot1, robot2, robot3: florero
    robotF: fiscalizador
comenzar
    AsignarArea(robot1,areaC)
    AsignarArea(robot1,areaPC)
    AsignarArea(robot1,area1)
    AsignarArea(robot2,areaC)
    AsignarArea(robot2,areaPC)
    AsignarArea(robot2,area2)
    AsignarArea(robot3,areaC)
    AsignarArea(robot3,areaPC)
    AsignarArea(robot3,area3)

```

AsignarArea(robotF,areaC)

AsignarArea(robotF,areaF)

Iniciar(robot1,1,1)

Iniciar(robot2,2,1)

Iniciar(robot3,3,1)

Iniciar(robotF,4,1)

fin

Ejercicio 5.

Existe un robot servidor que tiene su bolsa con papeles. Tres robots clientes tienen 4 intentos cada uno para solicitar al servidor que les entregue papeles. Cada vez que el servidor recibe un pedido de papeles de un cliente, se ubicará en la esquina (100,1), colocará allí una cantidad aleatoria de papeles (entre 1 y 5) y avisará al cliente correspondiente la cantidad de papeles que le depositó. Una vez que un cliente recibe un aviso, deberá recolectar uno a uno los papeles que le corresponden y depositarlos en su esquina inicial. El programa finalizará cuando todos los clientes hayan completado todos sus intentos. Asumir que el servidor tiene los papeles suficientes para cubrir todas las solicitudes. Los robots clientes inician en las esquinas (10,1), (11,1) y (12,1), respectivamente, y el robot servidor inicia en la esquina (13,1).

programa TP15_E5**areas**

```
areaC: AreaC(100,1,100,1)
area1: AreaP(10,1,10,1)
area2: AreaP(11,1,11,1)
area3: AreaP(12,1,12,1)
areaS: AreaP(13,1,13,1)
```

robots

```
robot cliente
```

variables

```
rob, avIni, caIni, papeles: numero
```

comenzar

```
avIni:=PosAv
caIni:=PosCa
RecibirMensaje(rob,robotS)
repetir 4
  EnviarMensaje(rob,robotS)
  RecibirMensaje(papeles,robotS)
  repetir papeles
    BloquearEsquina(100,1)
    Pos (100,1)
    tomarPapel
    Pos (avIni,caIni)
    LiberarEsquina(100,1)
    depositarPapel
```

fin

```
robot servidor
```

variables

```
rob, avIni, caIni, papeles: numero
```

comenzar

```
avIni:=PosAv
caIni:=PosCa
EnviarMensaje(1,robot1)
EnviarMensaje(2,robot2)
EnviarMensaje(3,robot3)
repetir 12
```

```
RecibirMensaje(rob,*)
Random(papeles,1,5)
BloquearEsquina(100,1)
Pos (100,1)
repetir papeles
  depositarPapel
Pos (avIni,caIni)
LiberarEsquina(100,1)
si (rob=1)
  EnviarMensaje(papeles,robot1)
sino
  si (rob=2)
    EnviarMensaje(papeles,robot2)
  sino
    EnviarMensaje(papeles,robot3)
fin
variables
robot1, robot2, robot3: cliente
robotS: servidor
comenzar
AsignarArea(robot1,area1)
AsignarArea(robot1,areaC)
AsignarArea(robot2,area2)
AsignarArea(robot2,areaC)
AsignarArea(robot3,area3)
AsignarArea(robot3,areaC)
AsignarArea(robotS,areaS)
AsignarArea(robotS,areaC)
Iniciar(robot1,10,1)
Iniciar(robot2,11,1)
Iniciar(robot3,12,1)
Iniciar(robotS,13,1)
fin
```