Ejercicios prácticos

Se dispone de la información de los participantes inscriptos a una carrera (a lo sumo 5000). De cada participante se tiene DNI, nombre y apellido, categoría (1..5) y fecha de inscripción. Se pide implementar un programa que guarde en una estructura adecuada los participantes de aquellas categorías que posean a lo sumo 50 inscriptos. Se sabe que cada participante se puede anotar en una sola categoría.
Idas para la solución:

- *La estructura que se dispone puede ser tanto una lista como un vector, porque sabemos la cantidad máxima de elementos, pero no la cantidad real*
- *Hay que ver cuántos participantes se incribieron por cada categoría. La información no está ordenada, con lo cual no es posible hacer un corte de control. La solución es usar un vector contador de 1 a 5, para contar los participantes por cada categoría*
- *Se debe recorrer una vez la estructura que se dispone, y calcular los totales en el vector contador (vc)*
- *Luego se debe volver a recorrer la misma estructura, evaluando if (vc[participante.categoria] <= 50) then agregarAdelante(listaResulante, vc)*
- *La estructura resultante debería ser una lista, agregando adelante o atrás (con puntero al último nodo). No tiene sentido aplicar un insertarOrdenado aquí. También se podría utilizar un vector con dimL en vez de una lista, aunque no parece tan conveniente (peor caso: 250 participantes, 50xcategoría).*

Realizar un programa que procese la información de los productos de un supermercado. De cada producto se lee código, rubro, stock y monto. La información se lee ordenada por rubro, hasta que llega el producto con código -1. Se pide almacenar, en una estructura de datos adecuada, solamente los productos del rubro “Limpieza”.

Solución general:

```
leerProducto(p)
//leo todos los productos que no son de limpieza
while  (p.codigo <> -1) and (p.rubro <> ‘Limpieza’) do
        leerProducto(p);

L := nil;
//ahora leo todos los que son de limpieza y los pongo en una lista
while (p.codigo <> -1) and (p.rubro = ‘Limpieza’) do
        agregarAdelante(L,p);
        leerProducto(p);
```

La Facultad de Informática organizará el congreso WICC 2022, en donde se expondrán trabajos de investigación. De cada trabajo se conoce: título, nombre del autor, DNI del autor y tipo de trabajo (1..10). Un mismo autor puede presentar varios trabajos. Se pide escribir un programa que:

a) Lea y almacene en una estructura adecuada la información de los trabajos. La lectura finaliza al ingresar un DNI con valor 22222222 (el cual debe procesarse).
b) Informar para cada autor la cantidad de trabajos presentados.

Ideas para la solución:

- *Para resolver el punto B, es necesario que la información esté ordenada por autor*
- *En el punto A, se debe insertar cada trabajo en una lista, ordenando por el DNI del autor. Se deberá utilizar una estructura repeat until como bucle para la lectura de trabajos.*
- *En el punto B habrá que hacer un corte de control, contando trabajos para cada DNI de autor*

Otra solución:

- *en vez de generar una lista de trabajos, se puede generar una lista de autores. Para cada autor se tendrá su DNI, cantidad de trabajos y una lista de trabajos.*
- *la lista de autores se armará nuevamente por DNI. Al leer un trabajo, si el DNI del autor no estaba en la lista, se crea un nuevo nodo, se copia el DNI y el nombre del autor, se coloca en 0 el campo "cantidad de trabajos", y se invoca a un módulo agregarAdelante con la lista de trabajos del autor (esta lista solo debería contener el titulo y tipo del trabajo). Si el DNI del autor sí estaba en la lista, se retorna el puntero a dicho nodo*

```
leerTrabajo(t);
buscarOCrearNodo(listaAutores,t, nodoDelAutor);
agregarAdelante( nodoDelAutor^.listaTrabajos, t);
nodoDelAutor^.trabajos := nodoDelAutor^.trabajos + 1;
```

Esta segunda solución mantiene toda la información de la solución anterior, pero evita datos repetidos. Además, para resolver el punto B, no hace falta hacer un corte de control ni tampoco evaluar todos los trabajos de la lista, sino simplemente evaluar todos los autores (que serán en el peor de los casos la misma cantidad de autores que trabajos).

Un comercio dispone de una estructura de datos con las facturas (como máximo 2000 facturas) realizadas durante enero de 2023. De cada factura se conoce el número de factura, código de cliente y monto total. También dispone de otra estructura con las facturas realizadas durante febrero (como máximo 2000 facturas). En ambas estructuras, las facturas se encuentran ordenadas por número de cliente. Se pide implementar un programa que incorpore las facturas del mes de febrero del cliente con código 444, de ser posible, a la estructura con las facturas de enero. Dicha incorporación debe realizarse de la manera más eficiente con respecto al tiempo de ejecución. Un cliente puede tener varias facturas por mes.

Ideas para la solución:

- *Ambas estructuras podrían ser tanto vectores (porque sabemos la cantidad máxima) como listas (porque no sabemos la cantidad real).*
- *Es conveniente utilizar listas, porque simplifica el código requerido para incorporar facturas de una estructura a la otra*
- *Se debe declarar un único tipo de dato "lista de facturas", y dos variables LE y LF para las facturas de enero y febrero respectivamente*
  La solución podría ser algo así (falta modularizar):

```
act := LE;
//posicionamos un puntero en la lista de enero, luego de la última factura del cliente 444
while (act <> nil) and (act^.dato.codcliente <= 444) do begin
  ant := act;
  act := act^.sig;
end;

//buscamos la primer factura del cliente 444 del mes de febrero
aux := LF;
while (aux <> nil) and (aux^.dato.codcliente < 444) do
  aux := aux^sig;
//copiamos las facturas de febrero a enero
while (aux <> nil) and (aux^.dato.codcliente = 444) do
        agregarAdelante(ant, aux^.dato);
        aux := aux^.sig;

//revisamos si hay que mover el puntero del inicio de la lista de facturas de enero
if (act = LE) then
        LE := ant;
```

Un comercio dispone de la información de sus ventas. De cada venta se conoce: código de producto, DNI del cliente, fecha, cantidad y precio unitario. La información está ordenada por DNI. Implementar un programa que genere una nueva estructura de datos que contenga DNI del cliente, cantidad total de compras que realizó y el monto total en compras.
Ideas para la solución:

- *Ambas estructuras (la de ventas y la de clientes) deben ser listas*
- *Se debe utilizar corte de control, calculando cantidad total y monto total por cliente*

```
procedure totalizar(L : listaVentas; var L2 : listaClientes);
var
  cli : cliente;
begin
        L2 := nil
        while L <> nil do
                cli.dni := L^.dato.dni;
                cli.monto := 0;
```

```
            cli.cant := 0;
            while (L<>NIL) and (cli.dni = L^.dato.dni) do
                  cli.cant := cli.cant + 1;
                  cli.monto := cli.monto + L^.dato.monto;
                  L :=  L^.sig;
            AgregarAdelante(L2,cli);
end;
```

Ejercicios de V ó F (y justificar)

No es posible la utilización de las variables globales para la comunicación entre los módulos de un programa. **Rta**: FALSO. Sí es posible, un módulo A puede asignar un valor a una variable global, y luego un módulo B puede acceder a dicho valor. Sin embargo, no se recomienda el uso de variables globales.

Siempre es posible realizar la eliminación de un elemento en un vector. **Rta**: FALSO. Para eliminar un elemento de un vector, deben verificarse algunas condiciones: que el vector no esté vacío, y que el elemento exista o la posición indicada sea válida.

Un programa modularizado puede no ser correcto. **Rta**: VERDADERO. El uso de módulos es recomendado, pero no asegura la corrección de un programa, ya que igualmente puede no cumplir con los requerimientos.

El acceso a un elemento de una estructura de datos lineal sólo es posible a través de un recorrido secuencial. **Rta**: FALSO. Un vector es una estructura de datos lineal, y puede accederse a sus elementos de manera directa utilizando un índice.

Las instrucciones dentro de una estructura de “repeat until” siempre se ejecutan. **Rta**: VERDADERO. La estructura de control Repeat Until primero ejecuta el bloque de instrucciones, y luego evalúa la condición, con lo cual se ejecutarán al menos una vez siempre.

Para que un programa sea eficiente debe estar modularizado. **Rta**: FALSO. Para el análisis de eficiencia de evalúa el tiempo de ejecución (evaluación teórica y empírica) y el uso de la memoria estática y dinámica. Un programa puede hacer buen uso de ambos recursos y no contener ningún módulo.

Un registro que posee todos sus campos del mismo tipo de dato es una estructura de datos heterogénea. **Rta**: VERDADERO. Un registro es una estructura de datos heterogénea siempre, ya que permite campos de distintos tipos.

Una estructura de datos vector puede estar almacenada en memoria dinámica. **Rta**: VERDADERO. Por ejemplo:

```
type
  vector = array[1..N] of integer;
  pvector = ^vector;
var
   v : pvector
begin
  new(v);
```

En el ejemplo anterior, si bien la variable v se almacena en memoria estática, luego de ejecutar la instrucción new podemos observar que lo apuntado por v es un vector, y se almacena en memoria dinámica.

Un elemento almacenado en una estructura de datos lista, puede estar en memoria estática. **Rta**: FALSO. Las listas (y todos sus elementos) se almacenan en memoria dinámica.

No es posible utilizar una estructura de control “for” para imprimir todos los elementos de una lista. **Rta**: FALSO. Si se conoce la cantidad de elementos de la lista, es posible recorrerla con un for. Por ejemplo:

```
for i:=1 to cant do begin
        writeln(L^.valor);
        l := l^sig;
end;
```

Una variable global puede ser pasada como parámetro real en la invocación a un módulo. **Rta**: VERDADERO. Si se trata de un parámetro por referencia, se recibirá dentro del módulo una referencia a la variable global; por el otro lado, si se trata de un parámetro por valor, se recibirá una copia del valor de dicha variable dentro del módulo.

Un pseudocódigo se realiza para un lenguaje de programación específico. **Rta**: FALSO. El pseudocódigo tiene como objetivo describir en alto nivel cuáles son las instrucciones que deberían ejecutarse, y se realiza de manera independiente de cualquier lenguaje de programación.

Un programa puede no contener una sección “type”. **Rta**: VERDADERO. Si no hace falta utilizar tipos de datos definidos por el usuario, entonces no se requiere una sección type.

En la invocación a un proceso, se puede pasar como parámetro real el valor constante 10. **Rta**: VERDADERO. Sólo se podrá pasar la constante si el parámetro es por valor; sin embargo, si se trata de un parámetro por referencia, entonces deberá pasarse una variable.

Una estructura CASE permite especificar múltiples casos verdaderos para el valor de la variable analizada. **Rta**: FALSO. La estructura CASE requiere la especificación de casos disjuntos. Si una evaluación es verdadera, entonces no habrá otra verdadera.

La técnica de corrección de debugging se aplica en la etapa del diseño de una solución. **Rta**: FALSO. La técnica de debugging se utiliza para buscar un error en el programa, y requiere haber superado la etapa de diseño y estar en etapas posteriores, como ser una etapa de implementación o una etapa de pruebas.

La comunicación mediante variables globales es más eficiente en cuanto al tiempo de ejecución. **Rta:** FALSO. El cálculo de tiempo de ejecución sólo considera operaciones aritméticas y lógicas, asignaciones y estructuras de control.

La invocación al módulo calcular es válida.

```
program ejercicio;
const max = 100.50;
var a, b: real;
  procedure calcular (var x: real; c: real);
  begin
    ...
  end;

begin
  ...
  calcular (max, 40);
  ...
end.
```

Rta: FALSO. El primer parámetro del módulo cálcular (max) debe ser una variable, que podrá ser modificada dentro del módulo mediante la referencia x. Sin embargo, se está pasando una constante, que no puede modificarse.

Siempre es posible eliminar un elemento de una posición determinada en un vector. **Rta**: FALSO. Si la posición indicada no es válida, ya sea porque supera la dimensión lógica del vector, o porque se encuentra fuera de los límites mínimo y máximo, entonces no podrá

eliminarse.

Todos los módulos “procedimiento” devuelven uno o más valores. **Rta**: FALSO. Un procedimiento puede no retornar ningún valor.

Para agregar un elemento e en una posición pos de un arreglo a sólo debo verificar que la dimensión lógica sea menor a la dimensión física del arreglo. **Rta**: FALSO. También se debe verificar que la posición sea válida (por ejemplo, que sea mayor o igual al índice inferior del arreglo).

La estructura de control for siempre puede ser reemplazada por la estructura de control while. **Rta**: VERDADERO. Tenemos dos casos:

Caso 1: for i:=MIN to MAX do sentencias;

Puede reemplazarse por:

```
 i := MIN
 while (i <= MAX) do
   sentencias;  i := i + 1;
```

Caso dos: for i:= MAX downto MIN do sentencias;

Puede reemplazarse por:

```
  i := MAX
while i >= MIN do
  sentencias; i:= i - 1;
```

Siempre que se debe buscar un elemento en una lista se usará un while que contiene la siguiente condición:

```
while (pri <> nil) and (pri^.dato <> buscado) do
begin
        ….
end;
```

Rta: FALSO. Si la lista está ordenada, la segunda condición del while debería ser > ó <, según el criterio de orden (ascendente o descendente).

Dados los siguientes módulos, el módulo A consume más memoria estática que el módulo B. Suponer que la lista tiene n elementos. Asuma que los punteros ocupan 4 bytes, los enteros 4 bytes, y los booleans 1 byte.

```
type
  lista = ^nodo;
  nodo = record
    dato: char;
    sig: lista;
  end;
```

Procedure A (ok: boolean; l: lista);
var a: integer;
begin
end;
	Procedure B (var ok: boolean; var l: lista);
	var a: integer;
	begin
	end;

Rta: FALSO. El módulo A requiere 9 bytes (1 para el boolean, 4 para el puntero L, y 4 para la variable local a), mientras que el módulo B requiere 12 bytes (4 para el parámetro por referencia ok, 4 para el parámetro por referencia L, y 4 para la variable local a).

Para definir un subrango de caracteres numéricos, la siguiente declaración es válida:

```
type
  numeros = '0'..'10';
var
  nums: numeros;
```

Rta: FALSO. El valor '10' no corresponde a un caracter de la tabla ASCII.

En la búsqueda de un elemento en un vector ordenado, siempre es necesario considerar la dimensión física por si el elemento no existe.
Rta: FALSO. Siempre se debe tener en cuenta la dimensión lógica, no la física.

Se puede agregar un elemento al final de una lista sin necesidad de realizar un recorrido secuencial de la misma.
Rta: VERDADERO. Se puede mantener un puntero al último nodo de la lista, lo que evita recorrer toda la estructura al momento de agregar un elemento al final.

Un módulo que no utiliza parámetros no puede comunicarse con el programa principal o con otros módulos.

Rta: FALSO. Puede comunicarse mediante variables globales, y en caso de las funciones puede comunicar su resultado retornando un valor hacia el programa principal.

Un módulo función puede retornar el acceso a un nodo de una lista.
Rta: VERDADERO. El acceso a un nodo es un puntero a dicho nodo. El tipo de datos puntero es un tipo simple, y las listas pueden retornar tipos simples.

Se quiere realizar un módulo que reciba una lista de enteros y un valor para agregar a la lista. Al finalizar debe retornar verdadero si se pudo agregar el elemento, teniendo en cuenta que no se agregan valores repetidos, caso contrario retornará falso. Dicho módulo podría ser una función. **Rta:** FALSO. Si se hace con una función, la misma debería recibir como parámetro un puntero al primer nodo de la lista. Podría darse el caso que el elemento a agregar esté al inicio de la lista, con lo cual sería necesario modificar el inicio de la lista. Para que este cambio se vea reflejado en el módulo que invocó a este módulo, el parámetro debería pasarse por referencia, lo cual no está permitido, ya que las funciones sólo pueden recibir parámetros por valor.

Una variable de tipo puntero puede ocupar más de 4 bytes en memoria estática. **Rta**: FALSO. Un puntero siempre ocupa 4 bytes, independientemente del tipo de dato al que apunte.

No es posible utilizar variables globales en un programa modularizado que utiliza parámetros. **Rta**: FALSO. El uso de variables globales no depende de la existencia o no de parámetros, pueden combinarse parámetros con variables globales (aunque no se recomienda el uso de variables globales).

Para buscar un elemento en una lista se puede utilizar una estructura repetitiva “for”. **Rta**: FALSO. Si se utiliza una sentencia FOR, asumiendo que se conoce la cantidad de nodos de la lista, se estará recorriendo toda la lista siempre. Sin embargo, para buscar un elemento, lo correcto es detener el bucle una vez que el mismo ha sido encontrado, con lo cual se requiere una estructura precondicional (while).

Cuando se conoce la cantidad máxima de elementos que se tienen que almacenar siempre se debe utilizar una estructura de datos estática para almacenar dichos elementos. **Rta**: FALSO. Conocer la cantidad máxima de elementos a almacenar no es suficiente para determinar la estructura de datos a utilizar; es necesario conocer el tipo de operaciones que se realizará con dicha estructura para tomar esta decisión. Por ejemplo, si se almacenarán datos de manera ordenada, insertarOrdenado en una lista es más eficiente que en un vector, ya que no se requieren corrimientos. Pero si se realizarán muchas búsquedas en la estructura, un vector puede ser mucho más eficiente que una lista si se utiliza un algoritmo de búsqueda dicotómica.

Un parámetro pasado por referencia puede ocupar lo mismo que un parámetro pasado por valor. **Rta**: VERDADERO. Por ejemplo, una lista pasada por valor será una copia del puntero inicial, o sea 4 bytes, y pasada por referencia será una referencia (o sea un puntero) al puntero inicial, o sea 4 bytes también.

Un módulo puede tener declarados tipos propios. **Rta**: VERDADERO. Los módulos pueden declarar sus propios tipos internos, que sólo pueden ser utilizados dentro del mismo módulo y sus sub-módulos.

Si se quiere realizar un módulo que reciba una lista de enteros, e incremente cada elemento en uno y al finalizar retorne verdadero si pudo realizar el incremento o falso en caso contrario, ese módulo puede ser una función. **Rta**: VERDADERO. Si bien no es recomendable, ya que el módulo estaría "retornando" vos valores (la lista modificada y el boolean), puede realizarse con una función, ya que la lista se pasará como parámetro por valor y sólo se modificará su contenido pero no su estructura. Dado que su contenido se encuentra en memoria dinámica, estas modificaciones quedarán reflejadas una vez finalizada la ejecución de la función.

Un programa que sólo utiliza variables globales no requiere ser modularizado. **Rta**: FALSO. Los módulos sirven para organizar la solución de un problema en problemas menores, facilitan las tareas de mantenimiento y simplifican las tareas de debugging. El uso de parámetros en los módulos es una característica deseable, pero no es indispensable.

---

a. El siguiente programa es válido.

```
program ejercicio;
 const
    num = 25;
  function auxiliar(val:integer): integer;
   begin
     val:= val * val;
     auxiliar:= val;
   end;
  procedure calculo(c: integer; var b:integer);
   begin
     b:= b + c DIV 4;
   end;
```

```
var
 a, b:integer;
begin
  a:= 16;
  b:= 6;
  calculo(auxiliar(a),num);
end.
```

Rta: FALSO. El segundo parámetro formal del módulo calculo se recibe por referencia, pero en su invocación se pasa una constante como parámetro real.

La estructura de control IF siempre puede ser reemplazada por la estructura de control CASE. **Rta**: FALSO. La estructura de control sólo CASE puede evaluar expresiones sobre tipos de datos ordinales. Si la condición del IF posee un tipo no ordinal, como ser un real, no podrá reemplazarse con un CASE.

Cuando se conoce la cantidad total de elementos que se tienen que almacenar siempre se debe utilizar una estructura de datos estática para su almacenamiento. **Rta**: FALSO. Para seleccionar una estructura de datos, también debe tenerse en cuenta el tipo de operaciones que se realizarán sobre la misma. Por ejemplo, si se insertarán nuevos elementos de manera ordenada, insertar ordenado en una lista es más eficiente que en un vector dado que no requiere corrimientos.

Si p es una variable de tipo puntero, entonces es válida la siguiente instrucción: writeln(p^). **Rta**: FALSO. Esa instrucción sólo será válida y el tipo de datos al que apunta P puede imprimirse, pero si se trata de una estructura de datos (por ej. un registro o un vector) esa instrucción será inválida.

Antes de reservar memoria para una variable del tipo puntero, siempre se la debe inicializar en nil. **Rta**: FALSO. No es necesario asignar NIL a una variable si luego se le asignará una dirección de memoria.

Un tipo de dato registro deja de ser una estructura de datos de acceso secuencial. **Rta**: FALSO. Un tipo de datos registro es una estructura de datos de acceso directo, pero no es de acceso secuencial

---

Un módulo función no puede retornar un tipo de dato definido por el usuario. **Rta**: FALSO. Si el tipo de datos es de tipo simple (por ej. un subrango de enteros o de char), entonces la función podrá retornar un valor de dicho tipo.

Un programa correcto con 2 variables locales es más eficiente en cuanto a la memoria que un programa con 5 variables locales. **Rta**: FALSO. El número de variables no determina la cantidad de memoria requerida, sino el tipo de datos de las variables. Por ejemplo, una variable de tipo vector de mil enteros ocupará más memoria que 5 variables de tipo entero.

Las acciones incluidas en la estructura de control for siempre se ejecutan al menos una vez. Rta: FALSO. Para calcular la cantidad de veces que se ejecuta un for, debe calcularse LIMITE SUPERIOR - LIMITE INFERIOR + 1. Por ej: la instrucción FOR i:= X to Y do se ejecutará (Y-X+1) veces. Si esa expresión da como resultado cero o menos que cero, entonces el cuerpo del for no se ejecutará. Por ej:
for i:= 1 to 0 do ⇒ (0 - 1 + 1) => 0 veces- Las acciones del for no se ejecutarán

Ejercicios donde piden evaluar código o soluciones a un problema

2. **Suponga la siguiente situación**: Se desea almacenar en una estructura la información de los alumnos de CADP. De cada alumno se conoce apellido y nombre y su promedio. Se sabe que a lo sumo habrá 800 alumnos. Además, semanalmente el profesor quiere agregar al final de la estructura nuevos alumnos y quiere implementar un módulo que sea eficiente en cuanto al tiempo de ejecución. ¿Qué estructura de datos le conviene elegir al profesor para que el agregar los alumnos sea eficiente en cuanto al tiempo de ejecución? Considere que siempre hay lugar en la estructura. **JUSTIFIQUE.**

Opciones
Opción 1: el profesor debe elegir un vector de alumnos donde para cada alumno se conoce apellido y nombre y su promedio. **Opción 2**: el profesor debe elegir una lista de alumnos donde para cada alumno se conoce apellido y nombre y su promedio. **Opción3:** cualquiera de las dos estructuras (vector o lista) es indistinto.

Rta: Es posible evaluar el tiempo de ejecución de ambas soluciones.
Si el profesor utiliza una lista, para agregar un nuevo alumno al final deberá ejecutar el siguiente código:

```
new(aux);
aux^.sig := NIL; //agrega al final
aux^.dato := alumno;
if (pri = nil) then pri := aux
else ult^.sig := aux;
ult := aux;
```

Esta solución requiere **5 unidades de tiempo**.
Por el otro lado, si el profesor utiliza un vector, el código a ejecutar será:

```
dimL := dimL + 1; //siempre hay lugar, con lo cual no se verifica dimL < dimF
v[dimL] := alumno;
```

Esta solución demanda **3 unidades de tiempo**.
Por lo tanto, al profesor le conviene elegir la opción 1.

2. Dada la siguiente declaración y los siguientes procesos, indique para cada uno de los procesos si son correctos o no. **El objetivo es duplicar el salario de cada empleado**. ***Justifique*** *su respuesta*.

```
type
  empleado = record
      nroEmpleado: integer;
      salario: real;
  end;
  vectorEmpleados = array [1..500] of empleado;
```

```
plantaEmpleados = record
    v: ^vectorEmpleados;
    dimL: integer;
end;
```

A

```
procedure duplicar1(p: plantaEmpleados);
var
  i: integer;
begin
  for i:= 1 to p.dimL do
    p^.v[i].salario := p^.v[i].salario * 2;
end;
```

B

```
procedure duplicar2(p: plantaEmpleados);
var
  i: integer; monto: real;
begin
  for i:= 1 to p.dimL do begin
    monto := p.v^[i].salario * 2;
    p.v^[i].salario := monto;
  end;
end;
```

En la solución 1, el parámetro se está pasando por parámetro, con lo cual se trata de una copia de la variable original. Sin embargo, el vector de empleados (campo v) se encuentra en memoria dinámica, y la copia pasada por parámetro apuntará a la misma dirección de memoria en su campo v, con lo cual los cambios sobre dicho vector dentro del módulo permanecerán al finalizar el módulo. Sin embargo, en esta solución existe un error al acceder al puntero dentro del parámetro p. La expresión p^.v[i].salario es incorrecta, ya que p no es un puntero; el puntero es p.v , y la expresión correcta sería p.v^[i].salario . Por lo tanto, el procedure duplicar1 no es correcto.

En la solución 2, nuevamente se pasa el parámetro por valor, y ya sabemos que es correcto. En esta solución, se accede correctamente al campo salario dentro del vector en memoria dinámica. Dentro del bucle for, se almacena en la variable monto el doble del salario de cada empleado, y dicho valor es luego utilizado para actualizar el mismo elemento del vector. El módulo duplicar2 es correcto.

2. Dada la siguiente declaración y los siguientes procesos, indique para cada uno de los procesos si eliminan correctamente el primer nodo de la lista “L”. Justifique su respuesta.

```
type lista = ^nodo;
     nodo = record
              dato: char; sig: lista;
            end;
```

A

```
Procedure eliminar1 (var L: lista);
begin
  if (l <> nil) then begin
    dispose(l);
    l:= l^.sig;
  end;
end;
```

B

```
Procedure eliminar2 (var L: lista);
var act: lista
begin
  if (l <> nil) then begin
    l:= l^.sig; act:= l;
    dispose(act);
  end;
end;
```

La solución A no es correcta, ya que primero elimina el puntero al inicio de la lista, con lo cual pierde el acceso a toda la lista. Además, cuando ejecuta la instrucción l:=l^.sig el programa fallará, ya que L no apunta a nada debido al dispose de la línea anterior.
La solución B tampoco es correcta, ya que la asignación act:=L se realiza luego de haber avanzado al segundo nodo de la lista. Al hacer esto, nuevamente pierde el puntero al inicio de la lista, pero además elimina el segundo nodo.

3. Dada la siguiente declaración de tipos de datos y variables, justificar para cada sentencia numeradas son válidas o inválidas:

```
program ejercicio_3;
type
  cadena50 = string[50];
  cliente = record
    DNI: cadena50; ape_nom: cadena50;
  end;
  clientes = ^nodo;
  nodo = record
    dato: cliente; sig: clientes;
  end;
var
  c: cliente; cli: clientes; cli_esp: clientes;
```

```
begin
   1. read(c);
   2. new(c);
   3. cli := nil;
   4. new(cli);
   5. cli_esp := cli;
   6. dispose(cli);
   7. read(cli_esp^.DNI);
   8. write(cli_esp^.DNI);
end.
```

Rta:
La sentencia 1 no es correcta, ya que no se puede leer un registro directamente.
La sentencia 2 no es correcta, ya que no se puede hacer new sobre una variable de tipo registro
La sentencia 3 no es correcta, ya que el valor nil sólo puede asignarse a variables de tipo puntero.
La sentencia 4 es correcta, ya que cli es un puntero a nodo.
La sentencia 5 es correcta, ya que siempre es posible asignar dos variables del mismo tiempo.
La sentencia 6 es correcta, ya que la instrucción dispose requiere un parámetro puntero, y cli es un puntero a nodo.
La sentencia 7 es incorrecta, ya que el puntero cli_esp apunta a una dirección de memoria inválida. Este puntero apuntaba a la misma dirección de memoria de cli (sentencia 5), y dicha dirección de memoria fue liberada previamente (sentencia 6). Además, para acceder al campo DNI, debería utilizarse la expresión cli_esp^.dato.DNI
La sentencia 8 es incorrecta por los mismos motivos que la sentencia 7.

Para la siguiente situación: “dado un vector de punteros a registros de empleados con nombre y dni, se quiere realizar un módulo que retorne el puntero al empleado de nombre “Juan García” que seguro existe. ¿Cuál/es de la(s) siguiente(s) opción(es) es/son correcta(s)? **JUSTIFIQUE.**

Opción 1: El módulo podría ser un procedimiento con dos parámetros, uno por valor (vector de punteros a los empleados) y otro por referencia que retorne el puntero al empleado con nombre “Juan García”. **Opción 2**: El módulo podría ser una función con un parámetro por valor (vector de punteros a los empleados) y que retorne el puntero al empleado con nombre “Juan García”. **Opción 3:** El módulo podría ser una función o un procedimiento.

La opción 3 es correcta, ya que un puntero puede ser devuelto como parámetro por referencia en un procedure, o como valor de retorno en una función (un puntero es un tipo de datos simple). Sin embargo, cabe destacar que retornar un puntero a un registro en una función puede no ser una buena práctica, ya que si bien la función retorna un único valor de tipo simple, en realidad estaría devolviendo de manera indirecta un registro.

2.- Dada la siguiente declaración y los procesos A y B, indique para cada uno de ellos si elimina correctamente o no el primer nodo de la lista recibida. **Justifique su respuesta**.

```
type numeros = ^nodo;          lista = record
     nodo = record                       pri: numeros;
              dato: real;                 ult: numeros;
              sig: numeros;             end;
            end;
```

A

```
procedure eliminar1 (var l: lista);
var aux: numeros;
begin
  if (l.ult <> nil) then begin
    aux := l.pri;
    l.pri := l.pri^.sig;
    dispose(aux)
  end;
end;
```

B

```
procedure eliminar2 (l: lista);
var aux: numeros;
begin
  if (l.pri <> nil) then begin
    aux := l.pri;
    l.pri := l.pri^.sig;
    dispose(aux)
  end;
end;
```

Rta: Se asume que el campo ult apunta al último nodo de la lista, y pri al primero. Y si la lista está vacía, tanto ult como pri serán nil. El proceso eliminar1 elimina correctamente el primer nodo de la lista. Sin embargo, no evalúa que el nodo a eliminar, apuntado por pri, sea el mismo que apuntaba ult, lo que significa que la lista tenía un solo nodo (y en cuyo caso tanto pri como ult deberían quedar en NIL), con lo cual no es correcto.
La segunda solución pasa el parámetro L por valor. En este caso, el registro L será una copia del registro original. Si bien los campos pri y ult de dicho registro apuntarán a los mismos nodos de la lista, al eliminar el primer nodo y finalizar el módulo, el registro original quedará sin alterar, con lo cual su campo pri quedará apuntando a una dirección de memoria inválida (dado que fue liberada dentro del módulo). Por lo tanto, la solución B tampoco es correcta.

2.- Dada la siguiente declaración y los siguientes procesos, indique para cada uno de los procesos si son correctos o no. El objetivo es duplicar el contenido del último nodo de la lista. **Justifique su respuesta**.

```
type numeros = ^nodo;
     nodo = record
              dato: real;
              sig: numeros;
            end;

lista = record
          pri: numeros;
          ult: numeros;
        end;
```

A

```
procedure duplicar1 (var l: lista);
begin
  l.ult^.dato:= l.ult^.dato * 2;
end;
```

B

```
procedure duplicar2 (l: lista);
begin
  while (l.pri^.sig <> nil) do
     l.pri:= l.pri^.sig;
  l.pri^.dato:= l.pri^.dato * 2;
end;
```

Rta: La solución A no es correcta, ya que en caso que la lista esté vacía, la expresión l.ult^.dato generará un error (ult no apunta a nada). Por otro lado, sin bien no afecta la ejecución del módulo, cabe destacar que esta solución pasa el parámetro L por referencia de manera innecesaria.
La solución B tampoco es correcta. Nuevamente, si la lista está vacía, la expresión l.pri^.sig no funcionará y generará error.

Apunte sobre cálculo de memoria y tiempo de ejecución

Consideraciones generales para el cálculo de tiempo de ejecución:

- Las instrucciones new, dispose, read y write NO SUMAN tiempo de ejecución. Sin embargo, podrían tener expresiones que requieren tiempo de cómputo. Por ej: **new(v[i+1])** tomará una unidad de tiempo, para el cálculo de i+1
- Las expresiones lógicas moleculares requieren el tiempo de cada expresión lógica atómica y de los conector lógicos. Por ej:
    - if (a>(b+c)) AND (d < 40) then
      Requiere:
      Condición 1: (a>(b+c)) ⇒ 2 unidades de tiempo (cálculo de b+c, y evaluación >)
      Condición 2: (d<4) => 1 unidad de tiempo
      Conector lógico: AND => 1 unidad de tiempo
      Tiempo total: 2 + 1+ +1 = 4 UT
- La estructura de control FOR requiere 3n+2+n.cuerpo
    - Para calcular el valor de n, se calcula LIM_SUPERIOR - LIM_INFERIOR +1 .
      Ejemplos:
      for i:= 3 to 45 do… ⇒ **N = 45 - 3 + 1 = 43**

      for i := 40 downto 6 do ⇒ **N = 40 - 6 + 1 = 35**

- La estructura de control WHILE requiere (n+1).condición + n(cuerpo). Se calcula el peor caso. Ejemplos:

```
readln(n); i:= 10;
while (n > 30) and (i <> 0) do begin
  readln(n); i := i-1;
end;
```

  El peor caso estará cuando i llegue a 0. En total se realizarán 11 iteraciones. N = 11
  El cuerpo del for requiere 2 UT (resta y asignación : i:=i-1)
  La condición del WHILE requiere 3 UT (dos comparaciones simples, y un AND)
  Tiempo del while (n+1).condicion + n.cuerpo = (11+1).3 + 11.2 = 36 + 33 = 69 UT

- La estructura de control REPEAT..UNTIL requiere n.condicion + n.cuerpo . Mismas consideraciones del WHILE
- La estructura de control IF requiere: condición + maximo(if, else)

```
if (a <> 5) and (v[i+1] = 0) then
```

```
                writeln('encontrado')
            else a:= a - 4;
```

Condición:

a <> 5 ===> 1 UT

v[i+1] = 0 ===> 2 UT (1 UT para calcular i+1, otra UT para evaluar =)

AND ===> 1 UT

Total condicion: 4 UT

Tiempo del IF: 0 UT (writeln)

Tiempo del ELSE: 2 UT

Tiempo total: condicion + max(if,else) ===> 4 + max(0,2) =====> 4 + 2 =6 UT

Los cálculos pueden quedar dependiendo de N. Por ej:

```
readln(n);
while (n mod 2 <> 0) do begin
   readln(n);
   v[i] := v[i]+1;
end;
```

No sabemos cuántas veces se ejecutará el while (no sabemos cuándo se leerá un número par). Entonces la cant. de iteraciones será X (desconocido). El tiempo quedará:
(x+1).condicion + x.cuerpo = (x+1).(2) + x.(2) = 2x + 2 + 2x = 4x + 2
Observaciones:

- la condición requiere 2 UT: una para calcular el mod, y otra para calcular el <>
- el cuerpo requiere 2 UT: una para calcular la suma y otra para la asignación
- las instrucciones readln no requieren tiempo

Puede haber situaciones donde hay un bucle dentro de otro. En vez de utilizar siempre la letra N, es conveniente utilizar distintas letras para identificar la cantidad de repeticiones de cada bucle. Por ej:

```
readln(cant);
for i:= 1 to cant do begin
   aux := (v[i] > 0);
   while (aux) do begin
         v[i] := v[i+1]  + 2;
        aux := (v[i] > 0);
   end;
end;
```

Ya sabemos que el readln no cuenta. Vamos a calcular el for:
Fórmula: 3N + 2 + N.cuerpo
No sabemos cuanto vale CANT, con lo cual quedará en función de N.

El cuerpo del for es:
(asignación + comparación) + tiempo del WHILE = 2 + ((M+1). condicion + M (cuerpo))
no sabemos cuántas veces se repetirá el WHILE,, con lo cual quedará en función de M:

2 + ((M+1).1 + M.(3 + 2)) = 2 + (M + 1 + 5M) = **3 + 6M UT**

Entonces el FOR quedará:
3N + 2 + N.(3+6M) = 3N + 2 + 3N + 6NM = **6N + 6NM + 2 UT**

**** **** **** **** **** **** **** **** **** **** **** **** ****

Para el cálculo de memoria:

- No olvidar las constantes
- Para calcular la cantidad de elementos de un vector: índice superior - índice inferior + 1 . Por ejemplo:
  - v : array[4..76] of integer ⇒ 76 - 4 +1 = 73 elementos

3. Calcule e indique la cantidad de **memoria estática y dinámica** que utiliza el siguiente programa. ***Mostrar los valores intermedios para llegar al resultado y justificar*.**

```
program ejercicio3;
const dimF = 1000;
type
  rango = 1..dimF;
  cadena = string[30];
  empleado = record
    dni: integer;
    ape: cadena;
    nom: cadena;
    salario: real;
  end;
  vectorEmp = array [rango] of empleado;
  planta = record
    emps: ^vectorEmp;
    dimL: integer;
  end;
var
  p: planta; e: empleado; i: integer;
begin
  new (p.emps); p.dimL := 0;
  for i:= 1 to (dimF DIV 2) do begin
    read(e.dni); read(e.ape); read(e.nom);
    read(e.salario);
    e.salario := e.salario * 2;
    p.emps^[i] := e; p.dimL := p.dimL + 1;
  end;
end.
```

Char	1 byte
Integer	6 bytes
Real	8 bytes
Boolean	1 byte
String	Longitud + 1 byte
Puntero	4 bytes

Memoria estática:
constante dimF + variables (p + e + i)
dimF es de tipo integer: 6 bytes según la tabla

p es de tipo planta, que es un registro con dos campos: un puntero (4 bytes) y un entero (6 bytes): 10 bytes
e es de tipo empleado, que es un registro con un entero (6b), un real (8b) y 2 cadena (31b x 2 =62b): 76b
i es de tipo integer: 6 b
Total memoria estatica: 6 + 10 + 76 + 6 = 98 bytes

Memoria dinámica:

- hay un new fuera del for, que reserva memoria para un vector (emps es un puntero a vector).
- El vector tiene dimF elementos (dimF - 1 + 1 = dimF = 1000). Cada elemento del vector es un empleado (76b). O sea, el vector ocupa 76.000 bytes
- No hay otras instrucciones para solicitar o liberar memoria, con lo cual el programa ocupa **76.000 de memoria dinámica**.

Podríamos también calcular el tiempo de ejecución si lo piden:

Asignación + for = 1 + for
FOR: se ejecuta 500 veces (1 to 1000 DIV 2 ⇒ 1 to 500 ⇒ 500 veces)
Los read no cuentan.
2 UT de la asignación y multiplicación
1 UT de la siguiente asignación
2 UT de la suma y asignación
Cuerpo del for: 2 + 1 + 2 = 5
Formula del for: 3N + 2 + N.cuerpo = 3x500 + 2 + 500x5= 4002 UT
Resultado: 4003 UT

5. Teniendo en cuenta las referencias, calcule e indique la cantidad de **memoria estática, memoria dinámica** y e **tiempo de ejecución**. Muestre cómo se obtienen los resultados.

```
program ejercicio_5;
type
 cadena30 = string[30];
 categorias = 1..5;
 participante = record
   ape_nom: cadena30;
   categ: categorias;
    tiempo: real;
 end;
 vector = array [1..20] of ^participante;
var
p: vector; i:integer; c: categorias;
ayn: cadena30;
begin
  for i:= 1 to 10 do begin
    new(p[i]); read(c); read(ayn);
    p[i]^.categ:= c; p[i]^.ape_nom:= ayn;
    p[i]^.tiempo:=0;
  end;
  for i:= 10 downto 5 do
    dispose(p[i]);
end.
```

Referencia	
Char	1 byte
Integer	4 bytes
Real	8 bytes
Boolean	1 byte
String	Longitud + 1
Puntero	4 bytes

4. Realice el cálculo de la memoria estática y dinámica del siguiente programa.
   Referencias: Integer (4), real (8), char (1), boolean (1) y puntero (4).

```
program ejercicio4;
const dimF = 50;
type
  cadena10= string[10];

  emple = record
    ape_nom: cadena10;
    cargo: cadena10;
    sueldo: real;
  end;
  vector = array[1..dimF]of ^emple;
```

```
var
 v: vector; e: emple; i:integer; suma, max: real;
begin
 i:= 0; suma:= 0; read (max);
 repeat
    i:= i + 1;
    new(v[i]);
    read(e.ape_nom, e.cargo, e.sueldo);
    v[i]^:= e;
    suma:= suma + e.sueldo
 until (suma > max) or (i = dimF);
 while (i > 0) do begin
  v[i]^.sueldo:=v[i]^.sueldo + v[i]^.sueldo*0.20;
  i:= i - 1;
 end;
end.
```

5. Calcule el tiempo de ejecución para el programa del ejercicio 4).

3. Calcule e indique la cantidad de **memoria estática y dinámica** que utiliza el siguiente programa. **Mostrar los valores intermedios para llegar al resultado y justificar.**

```
program ejercicio3;
const dimF = 100;
type cadena21 = string[21];
     alumno= record
              ape_nom: cadena21;
              promedio: real;
             end;
     vector = array [1..dimF] of ^alumno;
     lista = ^ nodo;
     nodo = record
          datos: alumno;
          sig:lista;
     end;
var  v: vector; a: alumno; nota, dimL, suma, cant: integer;
     e: info; aux: lista;
begin
  read(a.ape_nom); dimL:= 0
  while (a.ape_nom <> 'ZZZ') and (dimL < dimF) do
   begin
     read(nota); cant:= 0; suma:= 0;
     while (nota <> -1) do begin
       suma:= suma + nota; cant:= cant + 1;
       read(nota);
     end;
     if (cant <> 0) then a.promedio:= suma/cant
                    else a.promedio:= 0;
     dimL:= dimL+1;
     new (v[dimL]);
     v[dimL]^:= a;
     read(a.ape_nom);
   end;
```

Char	1 byte
Integer	4 bytes
Real	8 bytes
Boolean	1 byte
String	Longitud + 1 byte
Puntero	4 bytes

5. Indique la cantidad de memoria estática y dinámica que utiliza el siguiente programa y su tiempo de ejecución. Mostrar los valores intermedios para llegar al resultado. Referencias: Integer (6), real (8), char (1), boolean (1), puntero (4) y string (long + 1).

```
program ejercicio5;
type
  cadena19= string[19];
  alumno = record
    ape_nom: cadena19; edad: integer; sueldo: real;
  end;
  vector = array [1..10] of ^alumno;
var
 v: vector; a: alumno; i:integer;
begin
  for i:= 4 to 9 do begin
    new (v[i]);
    read(a.ape_nom, a.edad, a.sueldo);
    v[i]^ := a;
  end;
  for i:= 8 to 9 do
    dispose (v[i]);
end.
```

3.- Calcule e indique la cantidad de **memoria estática y dinámica** que utiliza el siguiente programa. **Mostrar los valores intermedios para llegar al resultado y justificar.**

```
program ejercicio3;
const dimF = 500;
type cadena = string[31];
  notas = 0..11;
  estudiante = record
    ape_nom: cadena;
    promedio: real;
  end;
  vector = array [1..dimF] of ^estudiante;
var v: vector; e: estudiante; nota: notas; i, j, suma, cant:
integer;
begin
  for i:= 1 to dimF do begin
     read(e.ape_nom); cant:= 0; suma:= 0;
     read(nota);
     while (nota <> 0) and (nota <> 11) do begin
       suma:= suma + nota;
       cant:= cant + 1;
       read(nota);
     end;
     if (cant <> 0) then e.promedio:= suma/cant
                    else e.promedio:= 0;
     new(v[i]);
     v[i]^:= e;
  end;
end.
```

Char	1 byte
Integer	4 bytes
Real	8 bytes
Boolean	1 byte
String	Longitud + 1 byte
Puntero	4 bytes

4.-Calcule el tiempo de ejecución del programa del punto 3. **Mostrar los valores intermedios para llegar al resultado y justificar.**

Qué imprime

2. Indique y justifique qué se imprime en cada sentencia write:

```
program ejercicio2;

procedure calcular (var b: integer; a: integer);
begin
  while ( b > 0 ) do begin
    a:= a + b;
    b:= b - 2;
    c:= c - b;
  end;
  writeln ('a= ', a, 'b= ', b,'c= ', c);
end;
var
  a, b, c: integer;
begin
 a:= 10; b:= -10; c:= 20;
 writeln ('a= ', a, 'b= ', b,'c= ', c);
 calcular (a, b);
 writeln ('a= ', a, 'b= ', b,'c= ', c);
end.
```

V ó F: El siguiente programa muestra por pantalla: `Valor a: 200 Valor b: 20 Valor c: 10`

```
program imprimir;
var a, c: integer;
procedure calcular (b: integer; var x: integer);
begin
 x:= 10; c:= c + b;   a:= (b + x) * 5;   b:= (a + b) MOD 10;
end;
var  b:  integer;
begin
 a:= 15; b:= 20; c:= b- a;
 calcular (b,c);
 writeln('Valor de a: ', a, '  Valor de b: ', b, ' Valor de c: ', c);
end.
```

3.- Dado el siguiente programa indique que imprime en cada sentencia write. **Justifique su respuesta.**

```
program tres;
var c: integer;

  procedure numero (d: integer; var b:integer; var c:integer);
  var a: integer;

  begin
    b:= (36 MOD 2) + d;
    a:= b + 3 * c;
    if (a >= 14) then b:= b + a * 2
                 else b:= b + a * 3;
    c:= a + b + c;
    writeln ('Valor a: ', a);
    writeln ('Valor b: ', b);
    writeln ('Valor c: ', c);
  end;

var a, b: integer;
begin
  a:= 4;  b:= 3;
  numero (b, c, a);
  writeln ('Valor a: ', a);
  writeln ('Valor b: ', b);
  writeln ('Valor c: ', c);
end.
```

Dado el siguiente programa indique que imprime en cada sentencia write. **Justifique su respuesta.**

```
program tres;
var c: integer;

  procedure numero (var a: integer; var b:integer; c:integer);
  var a: integer;
  begin
    b:= (18 DIV 4) + c;
    a:= b + 3 * c;
    if ((a + b) > 5) then b:= b + a * 2
                     else b:= b + a * 3;
    c:= a + b + c;
    writeln ('Valor a: ', a, 'Valor b: ', b, 'Valor c: ', c);
  end;

var a, b: integer;
begin
  a:= 4;  b:= 3; c:= 8;
  numero (b, c, a);
  writeln ('Valor a: ', a, 'Valor b: ', b, 'Valor c: ', c);
end.
```

Indique qué se imprime en cada sentencia writeln. Justifique.

```
program ejercicio;
var a, b : integer;
procedure calculo (c : integer; var b : integer);
begin
  b := a + c DIV 4;
  c := b MOD 3;
  a := b + c;
  writeln(a, b, c);
end;
var
  c : integer;
begin
  a := 10; b:= 6; C := 5;
  calculo(b, a);
writeln(a, b, c);
end.
```