Este apunte contiene secciones de preguntas teóricas y ejercicios prácticas provistos por la cátedra durante el curso de apoyo para rendir final de CADP, hay algunas preguntas que son para reflexionar y otras más de investigar, yo soy un alumno que escribió las respuestas mientras preparaba el final para usarlo de apunte, lo hice con mucho cuidado pero si tenés dudas sobre algo o no lo entendés investigalo más, suerte en el final!

Resolución de Problemas. Conceptos y definiciones iniciales

¿Qué es la Informática?

Ciencia que estudia la resolución de problemas del mundo real mediante el uso de una computadora.

¿Cuáles son las etapas en la resolución de un problema?

1. Obtener un problema
2. A partir del problema, obtener un modelo
3. Modularizar
4. Realizar el programa
5. Utilizar la computadora

¿Cómo especificarías las tareas involucradas en cada etapa?

- Obtener un problema y hacerse preguntas sobre el
- A partir del problema obtenido hacer un modelo del mismo
- Descomponer el problema en partes para encontrar una solución
- Diseño de su implementación, escribir el programa y elegir los datos a representar
- La computadora es capaz de aceptar datos de entrada, ejecutar con ellos calculos aritméticos y lógicos y dar información de salirda (resueltos), bajo control de un programa previamente almacenado en memoria

¿Qué es un algoritmo? ¿Qué es un programa? ¿Cuáles son las partes que componen un programa?

- Un **algoritmo** es un conjunto de pasos (instrucciones) a realizar sobre un autómata para obtener un resultado deseado en un tiempo finito
- **Programa** = Algoritmos + Datos
- Las instrucciones o acciones representan las operaciones que ejecuta la computadora al interpretar un programa. Un conjunto de instrucciones forma un algoritmo

  +

  Los datos son los valores de información de los que se necesita disponer y a veces transformar para ejecutar la función del programa.

¿Qué es una Precondición y una postcondición? ¿Como ejemplificaría cada una de ellas?

Pre-condición: es la información que se conoce como verdadera antes de iniciar el programa (o módulo)
Por ejemplo, en una función división las precondiciones son que los parámetros son números, y que el divisor sea distinto de 0. Tener una precondición permite asumir desde el código que no es necesario lidiar con los casos en que las precondiciones no se cumplen.

Post-condición: es la información que debería ser verdadera al concluir el programa (o módulo), si se cumplen adecuadamente los pasos especificados
En el ejemplo anterior, la función división con las precondiciones asignadas, puede asegurar que devolverá un número correspondiente al cociente solicitado.

Estructuras de Control

¿A qué llamamos estructura de control?

Estructuras que permiten modificar el flujo de ejecución de las instrucciones de un programa

¿Qué estructuras de control conoces?

Secuencia - Una sucesión de operaciones en la que el orden de ejecución coincide con el orden de aparición físico de las instrucciones
Iteración - Se ejecuta un bloque de acciones 0, 1 o más veces dependiendo de la evaluación de una condición.
Decisión - Toma decisiones en función de datos del problema.

¿Cómo clasificarías las estructuras de control que conoces en repetitivas, iterativas precondicional y postcondicional? ¿Cómo diferencias estos conceptos de estructura de control repetitiva, iterativa precondicional e iterativa postcondicional?

Las iterativas pre condicionales son cuando la condición se evalúa antes de ejecutar el bloque, mientras que la postcondición la ejecución se evalúa después de ejecutar el bloque de acciones.

¿Cómo implementarías la estructura repetitiva utilizando la estructura iterativa precondicional y postcondicional? ¿Hay alguna situación para la que la solución presente dificultades?

Precondicional : Un claro ejemplo es la sentencia While, mientras una condición sea verdadera se ejecuta un bloque de instrucciones.
Post condicional : Por ejemplo, Repeat Until, se ejecuta un bloque y mientras una condición no se cumpla volverá a ejecutarse.

La solución post-condicional presenta dificultad cuando se desea evaluar la condición antes de ejecutar la iteración, dado que el bloque se ejecutará por lo menos una vez antes de evaluar la condición

¿Cómo escribirías una forma equivalente de la estructura de control FOR utilizando la estructura de control WHILE? Y de forma contraria, es decir a partir de WHILE escribir una equivalencia con FOR, como lo harías?

por ejemplo, si quisiera escribir for i:=1 to 10 con la estructura while:
i := 1; while (i <= 10) do begin Accion; i := i +1; end;
No siempre podría escribirse una equivalencia a un while con un for ya que el for requiere saber a priori cuántas veces debe ejecutarse el código, mientras que el while no.

Datos y Tipos de Datos. Alcance de los datos

¿Qué es un dato?

Es una representación de un objeto del mundo real mediante la cual podemos modelizar aspectos del problema que se quiere resolver con un programa sobre una computadora. Puede ser variable o constante

¿Cómo diferencias el concepto de variable y el de constante?

Se diferencian en que las variables pueden cambiar su valor durante el programa y las constantes NO

¿A qué llamamos variables globales y variables locales?

Las variables globales son aquellas que podemos utilizar en todo el programa, incluyendo en los módulos; mientras las variables locales sólo pueden usarse dentro del módulo en que fueron declaradas

Cuáles de los siguientes conceptos se ven afectados cuando se utilizan variables globales:

a) Protección de los datos.

b) Cantidad de memoria utilizada.
c) Tiempo de ejecución del algoritmo.
d) Corrección de programa
e) Cantidad de líneas de código del programa.
f) Mantenimiento posterior del programa.
Justificar en cada caso.

a) Al utilizar variables locales estamos protegiendo al programa de que los módulos no alteren algún valor no deseado.
b) En las variables globales el espacio utilizado suele ser menor que declarar varias variables locales dentro de cada subrutina, por otra parte al utilizar variables locales el espacio se libera al finalizar el módulo
c) No se ve afectado
d) Puede afectarse, al utilizar variables globales en lugar de parámetros se corre el riesgo de alterar valores y que el programa no funcione como debería
e) No habría un cambio notorio
f) Es mucho más fácil mantener un programa que utilice variables locales como corresponde, ya que al hacer cambios en una variable global se debe contemplar que no traiga problemas en todos los módulos

¿Cómo definirías el concepto de tipo de dato? ¿Por qué crees que es útil tener tipos de datos?

Es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos, se caracterizan por:

- un rango de valores posibles
- un conjunto de operaciones permitidas
- una representación interna

Resulta útil para la gestión de la información

¿Qué diferencias hay entre los tipos de datos estándar de un lenguaje y los tipos definidos por el usuario?

El conjunto de valores, operaciones permitidas y representación interna de los tipos de datos estándar de un lenguaje están definidos y acotados por el lenguaje; un aspecto muy importante en los lenguajes de programación es la capacidad de especificar y manejar datos no estándar, indicando valores permitidos, operaciones válidas y representación interna. Un tipo de dato no estándar es aquel que no existe en la definición del lenguaje y el programador es el encargado de su especificación

¿Qué entiendes por ocultamiento y protección de datos? ¿Cómo defines el concepto de alcance de los datos?

Ocultamiento de datos significa que los datos exclusivos de un módulo no deben ser “visibles” o utilizables por los demás módulos.
Protección de datos se refiere a evitar que otros módulos no alteren un valor no deseado
El **alcance** es una propiedad de las **variables**: se refiere a su visibilidad (aquella región del programa donde la **variable** puede utilizarse). Los distintos tipos de **variables**, tienen distintas reglas de **alcance**.

¿Cómo vinculas los tres conceptos anteriores? ¿Podrías detallar cuáles serían los problemas de los lenguajes de programación que no implementen estos conceptos?

Están muy vinculados, el ocultamiento de datos tiene como objetivo proteger los datos y el alcance de ellos es donde pueden utilizarse
Al permitir que los datos puedan ser accedidos desde cualquier módulo se dificulta el trabajo en equipo ya que diferentes programadores tendrían que ponerse de acuerdo en que variables no deben usar en sus módulos y cuales si, además sería difícil utilizarlo y mantenerlo ya que se debería contemplar los efectos que tendría en todo el programa al momento de utilizar una variable

Modularización. Parámetros. Posibilidades en Pascal

¿Cómo defines el concepto de modularización? ¿Cuáles son sus principales características y ventajas?

Modularizar significa dividir el problema en partes funcionalmente independientes, que encapsulan operaciones y datos
Ventajas: Mayor productividad, Usabilidad, Facilidad de mantenimiento, Facilidad de crecimiento, Legibilidad.

¿Cómo diseñarías una solución modularizada adecuadamente? ¿Qué consideraciones tendrías en cuenta al descomponer en módulos?

Separaría el programa en funciones lógicas con datos propios y datos de comunicación perfectamente especificados.

- Cada subproblema está en un mismo nivel de detalle.
- Cada subproblema puede resolverse independientemente
- Las soluciones de los subproblemas pueden combinarse para resolver el problema original

¿Cómo relacionarías la modularización con la reusabilidad y el mantenimiento de los sistemas de software?

La descomposición funcional que nos ofrece la modularización favorece el reuso del software desarrollado.
La división lógica de un sistema en módulos permite aislar los errores que se producen con mayor facilidad, esto significa corregir los errores en menos tiempo y disminuye los costos de mantenimiento del sistema.

¿Cómo defines el concepto de módulo? ¿Qué módulos reconoce Pascal? ¿Qué diferencias hay entre dichos módulos?

Modulo: tarea específica bien definida, se comunican entre si adecuadamente y cooperan para conseguir un objetivo común.
En pascal existen los módulos PROCEDURE y FUNCTION
La diferencia radica en que los Procédure devuelven **0, 1 o más** valores de **cualquier tipo**; mientras las Function devuelve **un único valor** de tipo **simple**

¿Qué ventajas puede tener declarar tipos y variables dentro de los módulos?

- Reusabilidad
- Mantenimiento
- Se liberaría el espacio que ocupa en memoria una vez finalizado el módulo
- Me aseguro de que otros módulos no utilicen las variables

¿Que desventaja le encontrarías a un programa en el que no se han diseñado módulos?

- Si es muy grande sería imposible reutilizarlo
- Difícil de mantener
- Difícil de leer
- ..

¿Todo procedimiento puede ser transformado en una función? ¿Por qué?

No, si bien es posible en algunos, los procedimientos que devuelven más de 1 valor no pueden transformarse en función (estos solo pueden devolver un valor de tipo simple)

¿Toda función puede ser transformada en un procedimiento? ¿Por qué?

Si, porque siempre puede reemplazarse el valor de retorno con un parámetro por referencia

¿A qué llamamos bloque de programa?

Un bloque de programa es un conjunto de instrucciones que se ejecutan una detrás de otra

¿Cuáles son los mecanismos de comunicación entre módulos en Pascal y qué ventajas/desventajas ofrece cada uno? ¿Cuál es el más conveniente desde el punto de vista de la protección de los datos?

Los mecanismos son las variables globales y los parámetros

Las variables globales tienen como ventaja el hecho del espacio en memoria utilizado, el cual suele ser menor que declarar varias variables locales dentro de cada subrutina, sin embargo si hay muchos módulos que utilicen la misma variable puede terminar volviéndose dependientes unos de otros.
Problemas:

- Demasiados idenfiticadores
- No se especifica la comunicación deseada entre módulos
- Conflictos de nombres de identificadores utilizados por diferentes programadores
- Posibilidad de perder la integridad de los datos

Los parámetros favorecen la integridad de datos al declarar algunas variables como locales a un módulo asegurándose que no puedan ser visibles por otros módulos; así se especifican los datos que se comparten para comunicarse con los demás módulos.

Para proteger los datos es conveniente la parametrización, ya que utilizando variables globales puede modificarse datos de una variable involuntariamente en un módulo que luego deberá utilizar otro módulo.

¿A qué llamamos parámetros por valor y referencia?

Los parámetros por valor es llamado parámetro IN y significa que el módulo recibe (sobre una variable local) un valor proveniente de otro módulo o del programa principal
Con el puede realizar operaciones y/o cálculos pero no producirá ningún cambio ni tendrá incidencia fuera del módulo

La comunicación por referencia (out, inout) significa que el módulo recibe el nombre de una variable (referencia a una dirección) conocida en otros módulos del sistema.

Puede operar con ella y su valor original dentro del módulo, y las modificaciones que se produzcan se reflejan en el resto de los módulos que conocen la variable

¿Qué diferencias hay entre un parámetro por referencia y una variable global?

Los parámetros por referencia necesitan ser especificados en el módulo donde se utilizarán; las variables globales se declaran solo en el programa principal.
Los parámetros por referencia sólo pueden ser utilizados en los módulos donde se haya especificado; las variables globales pueden ser accedidas por cualquier módulo.
Los parámetros por referencia son nombrados y referencian a una variable del programa principal mientras las variables globales se invocan con el mismo nombre de la variable para todo el programa.

¿Qué argumento puedes dar para justificar el hecho de que en las funciones todos los parámetros deberían pasarse por valor?

Por definición las funciones devuelven un único valor y este lleva el nombre de la función (en pascal) los valores que reciba para procesar la información no deberían ser devueltos después de modificarse

¿Los parámetros utilizados en el módulo tienen alcance local o global? ¿Por qué?

Tienen alcance local, ya que solo son visibles dentro del módulo donde fueron declarados

Las siguientes consignas han sido recopiladas de finales de mesas anteriores.

Resolución de problemas, dato y tipos de datos

1. Se lee una secuencia de 10 números enteros. Escribí un programa que informe la cantidad de dígitos mayores, menores e iguales a 4 del total de números, y para cada número la suma de todos sus dígitos. Además que informe el número con mayor cantidad de dígitos.
En este ejemplo tratá de detallar las etapas que se debieron cumplir para llegar a escribir el programa.
Etapa 1: Leer el enunciado y abstraer el problema.
Etapa 2: Generar un modelo.
Etapa 3: Separar el problema en subproblemas.
Etapa 4: Escribir el código.

```
program Ejercitacion1;
procedure descomponer(n : integer; var cant,total: integer);
begin
     cant := 0;
     total := 0;
     while (n > 0) do
     begin
          total := total + n mod 10;
          cant := cant + 1;
          n := n div 10;
     end;
end;
var
   i,iguales,menores,mayores,max,numeroMax,n,
   sumaDeDigitos,cantidadDeDigitos : integer;

begin
     max := -1;
     iguales:=0;menores :=0; mayores:=0;
     for i:= 1 to 10 do
     begin
          read(n);
          if (n < 4) then
             menores := menores + 1
          else
             if (n = 4) then
                iguales := iguales + 1
             else
                 mayores := mayores + 1;
          descomponer(n,cantidadDeDigitos,sumaDeDigitos);
          write(sumaDeDigitos);
          if (cantidadDeDigitos > max) then
          begin
             max := cantidadDeDigitos;
             numeroMax := n;
          end;
     end;
     write(menores,iguales,mayores,numeroMax);
end.
```

2. Se lee una sucesión de números enteros terminados con el 9999, se desea obtener:
a) La suma de todas las unidades.
b) La suma de los tres últimos dígitos de cada número.
c) El promedio de los números leídos.
Ejemplo: se leen los números 132, 4201, 372, 23025, 9999 (no se procesa)
La respuesta será a) 10, b) 6-3-12-7, c) 5546.

```
program ej2;
function ultimosTres(n : integer):integer;
var
   tot,i : integer;
begin
   tot := 0;
   i :=0;
   repeat
        tot := tot + n mod 10;
        n := n div 10;
        i := i + 1
   until (i = 3) or (n = 0);
   ultimosTres := tot;
end;

var
   cont, n, suma : integer;
begin
     suma := 0;
     cont := 0;
     read(n);
     while (n <> 9999) do
     begin
          suma := suma + n;
          write (ultimosTres(n));
          cont := cont + 1;
          read(n);
     end;
     write (suma,suma/cont);
end.
```

3. Escribí un programa que lea una secuencia de caracteres terminada en punto e informe cuántas palabras contienen exactamente tres vocales distintas.
Además informá cuáles son las vocales de cada palabra.

(Puede estar mal)

```
program ej3;
var
    c,i : char;
    vocalesEncontradas, vocales : set of char;
    palabras,cantVocales: integer;
begin
    palabras := 0;
    vocales := ['A','E','I','O','U','a','e','i','o','u'];
    repeat read(c) until (c <> ' ');
    while (c <> '.') do
    begin
        cantVocales := 0;
        vocalesEncontradas := [];
        while (c <> ' ') and (c <> '.') do
        begin
            if (c in vocales) then
            begin
                if not (c in vocalesEncontradas) then
                begin
                    cantVocales := cantVocales + 1;
                    vocalesEncontradas := vocalesEncontradas + [c];
                end;
            end;
            read(c);
        end;
        for i := 'a' to 'Z' do
            if (i in vocalesEncontradas) then
               write('Se encontró ', i);
        if (cantVocales = 3) then palabras := palabras + 1;
        repeat read(c) until (c <> ' ');
    end;
    write ('Palabras con 3 vocales: ',palabras);
end.
```

4. Se tiene una secuencia de palabras, terminadas en punto. Cada palabra está separada por un único blanco. Se lee además una palabra de teclado. Se quiere saber cuántas palabras de la secuencia son **parecidas** a la palabra leída. Donde **parecidas** significa que coincidan en letra y posición en 3 ó más caracteres.
Ejemplo: "solero sol soleado sal soltero asolado solo."
la palabra leída: sol
Cantidad de palabras parecidas a sol: 5.
(tampoco funciona)

```
program ej4;
var
   texto, leida : string;
   i,j : integer;
   tam,cont, palabrasParecidas : integer;
begin
     texto := 'solero sol soleado sal soltero asolado solo.';
     readln (leida);
     palabrasParecidas := 0;
     tam := length(leida);
     i := 1;
     while (texto[i] <> '.') do
     begin
          cont := 0;
          j := 1;
          while (texto[i] <> '.') and (texto[i] <> ' ') and (j < tam) do
          begin
               if (texto[i] = leida[j]) then cont := cont + 1;
               i := i + 1;
               j := j + 1;
          end;
          if (cont >= 3) then palabrasParecidas := palabrasParecidas + 1;
          if (j = tam) then
             while (texto[i] <> '.') and (texto[i] <> ' ') do
                   i := i +1;
     end;
     writeln(palabrasParecidas);
end.
```

5. Dado el siguiente programa: informar que imprime en cada caso.

program ejercicio;
var alfa,beta , gama, epsilon: integer;

procedure calcular (alfa: integer; var gama : integer; var beta: integer; var epsilon: integer)
begin
alfa:= beta +1;
beta:= alfa + 5;
gama:= beta + 15;
epsilon:= beta - gama;
write (alfa); write (beta); write (gama); write (epsilon);
end;

begin
alfa:= 13; beta:= 16; gama:= -6; epsilon:= 4;
calcular(epsilon,alfa,beta,gama);
write (alfa); write (beta); write (gama); write (epsilon);
end.

Solución:
Dentro del proceso al principio:
alfa = 4, gama = 13, beta = 16, epsilon = -6.
luego alfa = 17, beta = 9, gama = 24, epsilon = -13.

Estos últimos valores son los que se imprimirán DENTRO DEL MÓDULO

alfa, beta y gama tomarán los valores gama, beta, epsilon del módulo respectivamente según su posición, dado que son utilizados por referencia mientras que los cambios en epsilon no serán reflejados en el programa principal

Entonces, alfa imprimirá 24, beta imprimirá 9, gama imprimirá -13 y epsilon 4.

Estructuras de Datos. Definición. Clasificación.
¿Qué es una estructura de datos?

Es un conjunto de variables relacionadas entre si y que se puede operar como un todo bajo un nombre único.

¿Qué estructuras de datos conoce?

Registro, Arreglo, Lista, Arbol.

¿Cómo clasifica a las estructuras de datos que conoce de acuerdo a:
- **Alocación de memoria**
- **Linealidad**
- **Acceso**
- **Los datos que la componen**

Registro: Alocación estática, lineal (los campos ocupan posiciones contiguas en memoria), Acceso directo, Heterogénea
Arreglo: Alocación estática, Lineal, Acceso directo, Homogénea
Lista: Alocación dinámica, linea, acceso secuencial, Homogénea
Arbol: Alocación dinámica, no lineal, acceso secuencial, Homogénea

¿Cuándo se dice que una estructura de datos es dinámica? y ¿estática?

Es dinámica cuando la cantidad de elementos que contiene varía durante la ejecución del programa, en cambio estática la cantidad es fija

¿A qué llamamos estructuras de datos homogéneas y heterogéneas?

Es homogénea si los datos que la componen son del mismo tipo, en cambio heterogénea pueden ser de distintos tipos

¿A qué llamamos estructuras de datos lineales y no lineales?

Lineal cuando está formada por ninguno, uno o más elementos que guardan una relación de adyacencia donde a cada elemento le sigue 1 y le precede uno, solamente.
Una estructura es no lineal si para un elemento dado pueden existir 0, 1 o más elementos que la suceden y/o preceden.

¿Qué diferentes accesos a las estructuras conoces?

Acceso secuencial: para acceder a un elemento debo respetar un orden.
Acceso directo: puedes acceder a un elemento directamente.

Registro
¿Cómo define el tipo de dato registro?

Permite agrupar diferentes clases de datos en una estructura única, compuesto por campos

¿Cómo es el acceso a los campos?

Directo. registro.campo

¿Es importante el orden de los campos? ¿Qué tipos de datos pueden contener los campos?

No importa el orden de los campos, los campos pueden contener datos elementales o bien otras estructuras de datos

¿Una variable tipo puntero puede ser campo de un registro? Por qué.

Si, la verdad no encuentro entiendo a donde apunta esta pregunta.. supongo que lo importante es destacar que el puntero es un dato simple que apunta a una dirección de memoria, pero si fuera otra estructura como un vector tampoco habría problema

Mencione las operaciones permitidas para el tipo de dato registro.

Asignación, comparación, lectura/escritura.

Arreglos
¿Cómo defines la estructura de datos ARREGLO?

Es una estructura de datos compuesta de elementos que se almacenan consecutivamente en la memoria que permite accederlos por medio de una variable índice que da la posición del componente dentro de la estructura de datos

¿Cuáles son sus operaciones?

- asignación de contenido a sus elementos
- Lectura/ Escritura
- Recorridos
- Cargar datos en un vector
- Agregar elementos al final
- Insertar elementos
- Borrar elementos
- Buscar un elemento

¿Cómo describirías la operación de agregar, insertar y borrar un elemento de un vector?

agregar: Si hay lugar en el vector, colocar el nuevo elemento en la posición de la dimensión lógica + 1 y aumentar la dimensión lógica
insertar: Si hay lugar en el vector, desde ‘dimensión lógica’ down to ‘posición donde quiero insertar’ -> me guardo lo que hay en la posición anterior a la actual, luego coloco el nuevo elemento en la posición deseada y aumento la dimensión lógica

borrar: Desde 'la posición que quiero borrar' hasta dimensión lógica menos 1, asigno el elemento de la posición siguiente, luego disminuyo la dimensión lógica

¿A qué le llamamos dimensión lógica y física del arreglo?

Dimensión lógica: Se determina cuando se cargan los contenidos del arreglo, indica la cantidad de posiciones de memoria ocupadas con contenidos cargados desde la posición inicial
Dimensión física: Se determina al momento de la declaración y determina su ocupación de memoria. La cantidad de memoria no variará durante la ejecución del programa

¿De qué tipo de dato se pueden declarar los elementos y el índice de un arreglo?
Los elementos pueden ser de cualquier tipo de datos de alocación estática (entero, real, caracter, logico, string, registro, otro arreglo) mientras que el índice debe ser de un tipo caracter, entero, subrango.

¿Cómo es el acceso a sus elementos?

Los componentes pueden recuperarse en cualquier orden, indicando simplemente su posición por eso es una estructura de datos de Acceso Directo, como el acceso es a través de un índice también se la denomina indexada

¿Cómo se declara el tipo de dato arreglo en Pascal?

arreglo = Array[1..dimF] of tipo

Listas
¿Cómo defines la estructura de datos LISTA?

Es una colección de elementos homogéneos, con una relación lineal que los vincula, es decir que cada elemento tiene un único predecesor (salvo el primero) y un único sucesor (salvo el último)
Los elementos que la componen no ocupan posiciones consecutivas, es decir que pueden aparecer dispersos en la memoria, pero mantienen un orden interno

¿Cuáles son sus operaciones?

Crear una lista vacía, Agregar un elemento al principio, Agregar al final, Insertar un elemento en una lista ordenada, Recorrer, Acceder al K-ésimo elemento de la lista, Eliminar un elemento, Combinar 2 listas (merge)

¿Cómo se declara el tipo de dato lista en Pascal?

```
lista = ^nodo
nodo = record
        dato : dato;
        siguiente : lista;
end;
```

¿Qué análisis podes hacer del uso de la memoria al trabajar con una lista?

Si se supone igual cantidad de datos en un vector y una lista, se puede afirmar que los vectores son más económicos ya que las listas requieren espacio extra para los enlaces

Si no se conoce la cantidad que contendrá cada estructura, no se puede declarar fácilmente los vectores teniendo que reservar en memoria más espacio del que realmente se necesita, mientras que las listas solo utilizan el espacio que se necesita.

Nota: los ejercicios no están corregidos ni compilados.

1. Cargar un vector **de hasta** 100 números enteros. Implementar un módulo que elimine todas las ocurrencias de un determinado número que se recibe como parámetro en ese módulo.

```
program ej1;
type
    vector = array[1..100] of integer;

procedure eliminar(numero: integer; var v:vector; var dl : integer)
var
    i : integer;
begin
    i := 1;
    while (i < dl) do
        if (v[i] = numero) then
        begin
            for j := i to dl-1 do
                v[j] := v[j+1];
            dl := dl - 1;
        end;
        i := i + 1;
end;
```

3. Detallar (realizando la descripción de cada paso) cómo se realizaría la inserción de un valor entero dentro de un vector ordenado de 10 elementos enteros, considerando que si el vector ya está totalmente ocupado quedarán los 10 elementos de mayor valor.

```
program ej2;
const
    df = 100

procedure InsertarOrdenado(var v:vector;numero: integer; var dl:integer)
var
    i,j :integer;
begin
    i := 1;
    while (v[i] < numero) and (i < dl) do i := i+1; //Recorro el vector buscando la posición
    if (v[i] < numero) then //Si debo insertarlo
        if (dl < df) then //Si tengo espacio debo agregarlo
            begin
                dl := dl +1;
                for j := dl downto i do v[j] := v[j-1]; //muevo todos los elementos una posic
            end
        else //si no hay espacio y el numero es mayor entonces lo reemplazo por el ultimo
            v[dl] := numero;
end;
```

4. Se desea procesar información de personas anotadas en una maestría. De cada una se conoce el apellido, nombre, dirección y especialidad. Se pide generar una lista ordenada por apellido.

```
program ej4
{
type
    persona = record
        apellido : string;
        nombre : string;
        direccion : string;
        especialidad : string;
    end;
    lista = ^.nodo
    nodo = record
        d : persona;
        s : lista;
    end;
procedure insrtarOrdenado(pri : lista; p : persona)
var
    ant,act,nue : lista;
begin
    new(nue);
    nue^.d := p;
    act := pri;
    ant := pri;
    while (act <> nil) and (p.apellido < act^.d.apellido) do
    begin
        ant := act;
        act := act^.s;
    end;
    if (ant = act) then
        pri := nue
    else
        ant^.s := nue;
    nue^.s := act;
end;
procedure leerPersona(var p : persona)
begin
    read(p.apellido)
    if (p.apellido <> 'fin') then
    begin
        read(p.nombre);
        read(p.direccion);
        read(p.especialidad);
    end;
end;

var
    l,nue,aux :lista;p : persona;

begin
    l := nil;
    leerPersona(p);
    while (p.apellido <> 'fin')do
    begin
        insertarOrdenado(l,p);
        leerPersona(p);
    end;
end.
```

5. Un comercio dispone de las ventas realizadas para sus productos. De cada venta se conoce número de producto (1..300), cantidad vendida y nombre de producto. Además el comercio cuenta con una tabla con el precio por unidad de cada uno de los 300 productos. Se pide calcular e informar el nombre del producto con el cual el comercio obtuvo la menor ganancia.
Notas: las ventas están ordenadas por número de producto. Un producto puede ser vendido 0, 1 o más veces.

```
program ejRecorrerLista
{
type
    venta = record
        numero : (1..300);
        cantidad : integer;
        nombre : string;
    end;
    lista = ^.nodo;
    nodo = record
        d : venta;
        s : lista;
    end;
    table = Array[1..300] of real;
var
    nombreMin,nombreAct : string;
    gananciaMin,gananciaAct : real;
    l : lista;
    t : tabla;
begin
    while (l <> nil) do
    begin
        nombreAct := l^.d.nombre;
        gananciaAct := 0;
        while (nombreAct = l^.d.nombre) and (l <> nil) do
            gananciaAct := gananciaAct + l^.d.cantidad * t[l^.d.numero];
            l := l^.s;
        if (gananciaAct < gananciaMin) then
        begin
            gananciaMin := gananciaAct;
            nombreMin := nombreAct;
        end;
    end;
    write(nombreMin);
end.
```

Corrección

¿Cómo defines el concepto de corrección?

El grado en que una aplicación satisface las especificaciones y consigue los objetivos encomendados por el cliente.

¿Cuando consideras que un programa es correcto?

Un programa es correcto si se realiza de acuerdo a sus especificaciones

¿Cuáles técnicas conoce para medir corrección y cómo describiría cada una de ellas?

Test: proveen evidencias convincentes respecto a que el programa hace el trabajo esperado.

Verificación: es el proceso de analizar las postcondiciones en función de las precondiciones establecidas.

Walkthrough: Recorrer el programa ante una audiencia

Debugging: es el proceso de localización del error

¿Si un programa es correcto también es eficiente?

Se define eficiencia como una metrica de calidad de los algoritmos. Asociada con una utilizacion óptima de los sistemas de computo donde se ejecutará el programa, principalmente la memoria y el tiempo de ejecución empleado.
Un programa puede tener varias soluciones correctas, sin embargo el tiempo de ejecución y memoria de cada solución puede ser muy diferente

¿Un programa que no presenta errores de compilación es correcto?

No, un programa podría compilar y no hacer lo que se especifica

¿Si eliges una solución modularizada, esto te asegura que el programa es correcto?

No

¿Un programa correcto asegura eficiencia?

No

¿Un programa correcto asegura legibilidad?

No

¿Un programa bien documentado asegura corrección?

No

¿La adecuada utilización de variables locales y globales ayuda a la corrección de una solución?

No

¿Crees que existe una única solución correcta a un problema planteado?

No

¿Las **estructuras de datos elegidas** determinan que una solución sea **correcta** o no?

No

En todos los casos piensa los por qué.

Eficiencia
¿Cómo defines el concepto de eficiencia?

Se la define como la métrica de calidad de los algoritmos, asociada con una utilización óptima de los recursos del sistema de computo donde se ejecutará el programa, principalmente la memoria utilizada y el tiempo de ejecución empleado

¿Cómo puedes medir la eficiencia de una solución dada?

Calculando el espacio ocupado y el tiempo de ejecución

¿Existe una relación directa entre eficiencia y corrección? ¿Una solución eficiente es siempre correcta? Por qué.

Para que un algoritmo sea eficiente primero se debe haber comprobado que es correcto. Si, no podría ser eficiente sin primero ser correcta.

¿Podrías relacionar el concepto de Eficiencia con los siguientes ítems?

- Cantidad de líneas del programa
- Modularización
- Uso de variables locales
- Uso de determinadas estructuras de control
- Excesiva documentación de los algoritmos

- Cantidad de líneas del programa: No se puede establecer una relación directa
- Modularización: facilita el analisis de eficiencia, dado que puede calcularse el tiempo de ejecución de los distintos procesos uno a la vez, sin embargo una solución modularizada no es necesariamente más eficiente que una sin modularizar
- Las variables locales tienden a mejorar la legibilidad pero no hacen al programa más eficiente
- Las estructuras de control no afectan el tiempo de ejecución

- Un algoritmo bien documentado puede ser menos eficiente que otro

¿Cuáles crees que serían los ítems que influyen sobre el tiempo de ejecución de un programa y por qué?

a) Cantidad de datos de entrada
b) Cantidad de líneas de código
c) Cantidad de iteraciones presentes en el programa
d) Cantidad de variables declaradas

A. Si un programa se va a ejecutar solo con entradas “pequeñas” , la velocidad de crecimiento puede ser menos importante
B. No hay una relación directa
C. Las iteraciones multiplican el tiempo de ejecución del bloque
D. No lo afecta

Análisis comparativo de las estructuras de datos vistas

¿Qué diferencias conceptuales existe en las estructuras de datos vectores y listas?

Las dos estructuras son homogeneas
El acceso a los elementos en el vector es directo a través de un índice, en cambio en la lista es secuencial recorriendo los elementos de la lista
Las dos estructuras son lineales
Los vectores almacenan memoria continua en memoria y las listas se almacenan aleatoriamente, siguiendo un orden lógico
En vectores la ocupación en memoria se resuelve al momento de la **compilación** y en listas la ocupación en memoria se resuelve al momento de la **ejecución**
Los vectores poseen acceso directo, en cambio las listas son de acceso secuencial

Compara y explica detalladamente la operación de inserción y borrado en vectores y listas.

En los vectores al momento de insertar un elemento primero debe comprobarse que haya espacio, entonces se mueven todos los elementos desde la dimensión lógica hasta la posición donde se desea insertar el elemento, finalmente se asigna el elemento a la posición
En las listas no se controla el espacio en memoria, en cambio debe reservarse un lugar de alojado usando new() y generar el nuevo nodo, requiere menos accesos a memoria dado que al momento de la inserción simplemente se ajustan los punteros del nodo anterior (si no es el primero) y el puntero al siguiente elemento

Al borrar un elemento en un vector debe asignarse a cada elemento su siguiente desde la posición del elemento que deseo borrar hasta la dimensión lógica, finalmente reducir la dimensión lógica
Al borrar un elemento de una lista simplemente deben ajustarse los punteros para que el puntero anterior al nodo borrado (si no es el primero) apunte al nuevo nodo y el nuevo nodo apunte al de la posición actual

¿Cómo diferenciaría la búsqueda de un elemento en un vector y en una lista?

En los vectores hay que controlar que mientras se busca el elemento la dimensión lógica no exceda la dimensión física y avanzar aumentando una variable que sirva de índice
En las listas debe controlarse que no se llegue al último elemento de la lista y se recorre a través de los punteros al siguiente elemento, utilizando una variable auxiliar de tipo "lista"

¿Cómo es el acceso a un elemento conociendo su posición en un vector y en una lista?

En un vector el acceso es directo, vector[pos] mientras que en las listas el acceso es secuencial, es decir deben recorrer los componentes previos para llegar a la posición deseada

¿A igual cantidad de elementos en un vector y una lista qué diferencias hay respecto de la ocupación de memoria?

La diferencia es a favor de los vectores, ya que estos podrían almacenarse directamente en memoria en cambio en las listas se almacenará un puntero por cada componente además del elemento a guardar.

Eficiencia

1. a) Se desea almacenar información sobre los últimos 12 resultados del campeonato mundial de Basketball. Si tuviera que ir guardando esta información en una estructura de manera tal de minimizar la cantidad de memoria utilizada: Seleccione la opción más adecuada y justifique.

Un vector
Una lista simple enlazada

Un vector sería la estructura más eficiente, dado que se conocen a priori la cantidad de elementos que almacenará y que utilizando una lista ocuparía espacio extra con los punteros

b) Compare la eficiencia de las operaciones de agregar, eliminar y buscar un dato en una lista simple cuando la misma está ordenada y cuando no lo está.

agregar es más eficiente cuando la lista está desordenada, ya que si estuviese ordenada necesitaría mantener ese orden ajustando los punteros por cada vez que debo agregar un elemento; en cambio si deseara agregar en una lista desordenada siempre podría agregar al principio o al final
eliminar tiene la misma eficiencia tanto en una lista ordenada como una desordenada, dado que en ambos casos tendrá el mismo trabajo de ajustar los punteros
buscar un dato es más eficiente en una lista ordenada, ya que permite realizar un corte de control para no tener que recorrer la lista por completo si el elemento no está en ella

En conclución una lista ordenada será más eficiente que una desordenada cuanto menor cantidad de veces que se actualize y mayor cantidad de veces se consulte

2. a) Mejore la eficiencia del siguiente bloque de código y justifique:

```
B:= 2;
C := 3;
For I:= 1 to 2000 do
   X := 2 * (B + C) * I;
```

```
B:= 2;
C := 3;
D := 2 * (B + C)
For I:= 1 to 2000 do
   X := D * I;
```

```
D := 2 * (2 + 3)
For I:= 1 to 2000 do
   X := D * I;
```

```
D := 2 * (2 + 3)
X := D * 2000;
```

```
X := 2 * (2 + 3) * 2000
```

En la primer mejora dejo el cálculo de 2 * (B + C) fuera del for, debido a que mejora el tiempo de ejecución sin afectar lo que el código hace.
En la segunda mejora elimino las variables B y C para ocupar menos lugar en memoria, esto solo puede hacerse si no son relevantes para el resto del programa
En la tercer mejora quito el for, debido a que no importa lo que suceda durante las iteraciones el resultado final siempre será el mismo y me está multiplicando el tiempo de ejecución del bloque
En la cuarta me deshago de la variable D para ahorrar memoria

3. Suponga que se pide crear una estructura de datos con 100 combinaciones diferentes de tres letras. Interesa que cada una de esas combinaciones se forme al azar. Ud. dispone de un procedimiento TRESLETRAS, que le devuelve un dato de tipo string con tres letras de la A a la Z mayúsculas al azar.
a) Resuelva lo pedido invocando al procedimiento TRESLETRAS, verificando que esa combinación ya no haya sido generada y guardándola en la estructura. Cuando ya esté generada, no la debe guardar.
b) A partir de la solución propuesta, realice un análisis de la misma desde el punto de vista de la memoria utilizada y del tiempo de ejecución empleado (calculando cantidad de operaciones efectuadas). Nota: el procedimiento TRESLETRAS no hay que implementarlo, solo se lo invoca.
c) Con respecto al inciso (a), para el proceso de búsqueda en la estructura elegida, ¿utiliza un algoritmo de recorrido secuencial?
d) ¿Como procede con la incorporación de nuevos datos a la estructura?

```
1   program ejTresLEtras;
2   {
13  type
14      vector = array[1..100] of String;
15
16  function noEstaGenerada(v : vector;s : string; dl : integer): boolean
17  var
18      aux : boolean; i:integer; //memoria: +1 unidad boolean + 1 unidad integer
19  begin
20      aux := false;   i := 1; // 2 unidades de tiempo
21      while (i < dl) and (aux = false)do //(entre 0 y 100) * 2 unidades de tiempo
22      begin
23          if (v[i] = s) then aux := True  // 2 unidades de tiempo //aplicar regla del if
24          else i := i+1;  //2 unidades de tiempo
25      end;
26  end;
27  var
28      s : string; v : vector;     //memoria : +1 unidad de string , +100 unidades de string
29      cont, dl : integer; //memoria: + 1 unidad integer + 1 unidad integer
30
31  begin
32      dl := 0; // 1 unidad T
33      cont := 0; // 1 unidad T
34      TRESLETRAS(s);
35      while cont < 100 do // N unidades de tiempo
36      begin
37          if noEstaGenerada(v,s,dl) then //N * Tiempo de noEstaGenerada * 1 unidad de tiempo
38          begin
39              dl := dl +1; //N * 2 unidades de tiempo
40              v[dl] := s; //N * 1 unidad de tiempo
41              cont := cont + 1; //N * 2 unidades de tiempo
42          end;
43      end;
44  end;
```

Memoria: 1 byte + 4 bytes + 256 (cantidad de caracteres +1) + 100 * 256 + 4 bytes + 4 bytes
Tiempo de ejecución: 2 T + N * T + N * Tiempo de función * T + 5N * T

Si utilizo un algoritmo de recorrido secuencial ya que no tengo otra opción, si la estructura estuviera ordenada por algún criterio podría realizar una búsqueda dicotómica para reducir el tiempo de ejecución

Dado que la estructura es un vector y que se que mi programa no puede almacenar más de 100 elementos no necesito controlar que la dimensión lógica supere la física, simplemente aumento la dimensión lógica y coloco el nuevo elemento en su posición