Autoevaluación y guía de estudio para las unidades de pseudo-código

Esta guía pretende servir de autoevaluación para el alumno; no con el objetivo de evaluar su desempeño completo, sino solamente lo más básico y mínimo de cada unidad. Para cada una se presentan uno o dos problemas que al final de la unidad deberían resultar triviales (muy fáciles), ya que son de los ejemplos más sencillos posibles.

Debe resultar completamente obvio cómo se resuelve cada uno de estos problemas. Y esto no es recordar de memoria la solución o un ejercicio patrón, sino razonar el por qué de cada linea de código. Si esto no ocurre con un problema de los presentados aquí para alguna unidad, entonces el alumno aún tiene faltas conceptuales graves, y no debería avanzar a la siguiente unidad hasta no resolverlas (debería volver a leer/ver la teoría desde cero).

Luego del/de los problemas triviales, para cada unidad se discute brevemente cómo se complicarán en ejercicios más reales/de parcial y se dejan algunas recomendaciones para la práctica.

Recordemos que en esta materia los conocimientos se apilan unidad tras unidad, en cada nuevo tema estamos reusando como bases todos los anteriores, no son temas estancos o independientes. Entonces, resulta inútil pretender avanzar al siguiente sin tener claro lo básico.

Lo repetimos una vez más: **si a un alumno no le sale alguno de estos ejercicios marcados como *triviales*, entonces le falta entender algo muy básico de esta unidad y debe repasar su teoría desde cero; no debe intentar avanzar a la siguiente unidad porque será perder el tiempo**.

Unidad 2

Ejercicio trivial:

- Escriba un algoritmo que permita ingresar los dos lados de un rectángulo, y luego calcule e informe el área y el perímetro del mismo.

Conceptos básicos involucrados: Las tres primitivas, secuenciales: *leer escribir* y *asignar*; y de la mano de ello el concepto de *variable*, tal vez el más importante en esta unidad.

«asegúrese de poder resolver ese ejercicio sin tener que pensar más de 2 minutos, antes de seguir leyendo»

¿Cómo se puede complicar? Todos los problemas en esta guía son iguales: se leen los datos, se aplican las relaciones asignando los resultados en una o más variables, y finalmente se escriben los resultados. Desde la codificación no hay casi nada que se pueda complicar, la clave siempre es hacer un buen análisis antes. La complicación puede ver por la matemática del problema para encontrar las fórmulas/relaciones. Ej:

- Encontrar las raíces de una ecuación cuadrática: implica saber qué se entiende por raíces, conocer y aplicar la fórmula de la resolvente, encontrar coeficientes de prueba que no generen raíces imaginarias.
- Como **caso particular**, calcular la edad a partir de dos fechas: implica entender la ventaja de representar la fecha como un solo nro aaaammdd, y saber operar con el mismo (armarlo, desarmarlo, comparar), lo cual implica poder usar funciones como Trunc, u operadores ”nuevos” como el Mod. Esta forma de operar con fechas será útil en más de una oportunidad más adelante.

¿A qué prestar especial atención al codificar? En fórmulas no triviales, al uso de paréntesis para que la precedencia de operaciones sea la adecuada (por ej, en la resolvente, para que la división entre numerador y denominador no deje nada fuera).

Recomendación para la práctica: No omitir la etapa de análisis (tener bien definido el problema, e identificados datos, resultados y relaciones) antes de intentar codificar.

Unidad 3

Ejercicios triviales

- Escriba un programa que permita ingresar dos números y determine cuál es el mayor
- Escriba un programa que permita ingresar 30 número e informe el promedio de todos ellos.

Conceptos básicos involucrados: *estructura de control condicional* (Si-entonces) *y repetitiva* (Mientras o Repetir); y para poder utilizarlas, el planteo de *expresiones lógicas* mediante operadores relacionales (comparaciones).

«asegúrese de poder resolver esos ejercicios sin tener que pensar más de 2 minutos, antes de seguir leyendo»

¿Cómo se puede complicar? Los ejemplos triviales apuntan a una sola estructura de control cada uno, y a un cálculo o pregunta muy simple.

- La pregunta podría tener que ser más compleja: usar operadores lógicos (y,o) además de relacionales para ”combinar” varias preguntas en una (ej, determinar la condición de un alumno, libre, regular o promocionado, a partir de las notas de sus parciales y recuperatorios).
- Se podría requerir anidar estructuras de control (ej: el mayor de 3 anidando dos Si-entonces, o el mayor de N anidando un Si-entonces en un Mientras, o los dos mayores de N).
- El ”resultado” a calcular puede ser más complejo. En el caso de procesar muchos datos, una suma, un promedio o hasta un conteo, son casos simples. Buscar uno o los dos mayores/menores, o leer sin saber a priori la cantidad de datos son casos típicos y un poco más complejos desde la implementación. Algunos menos habituales como determinar si un número es primo son más complejos desde el análisis mismo del problema, además de la implementación.

¿A qué prestar especial atención al codificar? Lo primero y más importante es identificar bien cuándo hay que usar cada tipo de estructura; y asociado a esto cuándo anidar. En segundo lugar está plantear bien las condiciones, o los cálculos que van dentro. Lo segundo no deja de ser importante para el resultado, pero un error allí es más fácil de ver y corregir de a poco, mientras que los primeros (por ej, usar un Si-Entonces donde iría una repetitiva, o anidar dos repetitivas cuando no corresponda) son errores más conceptuales y que pesan mucho más en la nota.

Recomendaciones para la práctica: Se recomiendo en esta unidad utilizar el diagrama de flujo en lugar del pseudocódigo, ya que allí suele ser más fácil entender el uso y funcionamiento de las estructuras de control. Por otro lado, para los problemas que requieran procesar muchos datos, es útil ponerse en el lugar de la máquina (ver los datos uno por vez, nunca ver la lista completa) para intentar resolver uno mismo el problema en su cabeza y luego pensar cómo lo hizo.

Unidad 4

- Escriba un programa que permita ingresar una lista de 30 nombres y los guarde en un arreglo. El programa debe luego mostrar la lista nuevamente.
- Escriba un programa que permita cargar los 50 datos de una matriz de 10x5, por fila. El programa debe luego informar primero los datos de las 3ra columna. Luego, los de la 4ta fila.

Conceptos básicos involucrados: Solo la idea de *arreglo*, el resto es extensión de las unidades 2 y 3.

«asegúrese de poder resolver esos ejercicios sin tener que pensar más de 2 minutos cada uno, antes de seguir leyendo»

¿Cómo se puede complicar?

- Estrictamente pensando en arreglos, la mayor complicación se puede dar en la carga de datos, cuando estos no vienen en orden (la típica lectura ”por ternas”, donde además puede haber que acumular; ejemplo: ventas).
- La otra diferencia en los ejercicios no triviales será que luego de cargado el arreglo se pedirá calcular cosas más complejas, pero casi siempre serán ”cosas” que ya hemos calculado en la unidad 3 (mayores/menores, sumas, conteos, promedios, listados, etc).
- Como **casos particulares**, es importante detenerse en los algoritmo para buscar/insertar/eliminar un elemento de un arreglo, ya sí son ”cálculos” nuevos, no tenían sentido en la unidad 3.

¿A qué prestar especial atención al codificar? No mecanizarse pensando que cualquier operación sobre un arreglo 1d implica un Para, y sobre uno 2d implica dos Paras anidados. En cada caso analizar si realmente necesito recorrer todos los elementos, o solo algunos y en base a eso decidir cuándo y cómo utilizar las estructuras iterativas. Ejemplos: si debo mostrar o modificar el 5to elemento de un arreglo, no necesito iterar, es una sola operación y se el índice; o si quiero mostrar una fila de una matriz no necesito un doble Para, porque solo debe mover el índice de columna, pero el de fila queda fijo.

Recomendaciones para la práctica Es clave elegir bien la estructura de datos (cuándo usar arreglo, cuantas dimensiones, qué tamaños). Si la estructura de datos está bien elegido (y entendido el por qué/para qué), luego el código se hace algo más fácil/obvio. Lo más novedoso en estos ejercicios debería ser la carga de datos, pero luego los cálculos sobre los datos cargados son cosas que ya hicimos en la unidad 3. Por ejemplo, la idea del algoritmo para sacar el mayor de N números, es la misma sin importar si los N números se leen a medida que se necesitan (como en la unidad 3), si provienen de un arreglo 1D, o de una fila/columna de una matriz.