4. MEJORANDO EL DISEÑO DEL CÓDIGO

Es muy común encontrar algoritmos construidos de tal manera que hacen lo que se necesita pero codificados de una manera *que tiempo después ni el mismo autor sabe cómo lo hizo.* Sólo sabe que funciona. Y si se trata de modificarlo, ahí sí que menos posibilidades hay. Por esto es importante acostumbrarse desde el momento en que se está aprendiendo a programar, a tratar de escribir el código en la forma más clara y con la menor redundancia posible.

Ejemplo 1. **Mejore** el diseño del siguiente segmento de algoritmo. Justifique el nuevo diseño.

DISEÑO INICIAL	**NUEVO DISEÑO**
... si (a > b y a ≠ b) entonces p ← a k ← p+b escriba k si_no si (a ≤ b) entonces p ← a escriba k k ← p+b fin_si fin_si ...	**...** **p ← a** **si (a > b) entonces** **k ← p+b** **escriba k** **si_no** **escriba k** **k ← p+b** **fin_si** **...**

Justificación. ¿Qué significa (a > b y a≠b)? a≠b significa que a no es igual a b. Esto a su vez quiere decir que hay dos posibilidades: a < b ó a > b. Por lo tanto, (a > b y a≠b) se puede remplazar por (a > b y (a < b ó a > b)). Para que la evaluación de esta condición compuesta resulte verdadera se requiere que a > b sea verdadera y que (a < b ó a > b) sea verdadera. Obsérvese que la primera condición (a > b) fuerza la selección del resultado de la segunda (a < b ó a > b) porque para que el resultado entre a>b y otra condición sea verdadero, la única posibilidad es que la evaluación de las dos condiciones resulte verdadera. Al ser a > b ya no puede ser cierto que a < b. Así que de (a < b ó a > b) se selecciona a > b quedando

(a > b y a > b) lo que se equivale a decir que a >b simplemente.

En conclusión, (a > b y a≠b) se puede remplazar por a > b en aras de la claridad.

Hasta aquí se tendría la siguiente versión mejorada:

```
...
si (a > b) entonces
    p ← a
    k ← p+b
    escriba k
si_no
    si (a ≤ b) entonces
        p ← a
        escriba k
        k ← p+b
    fin_si
fin_si
...
```

Obsérvese ahora que para entrar a ejecutar el cuerpo del si_no se debe satisfacer la condición a ≤ b (lo contrario de a > b) por lo tanto, aunque no hace daño, sobra preguntar si a ≤ b.

Teniendo en cuenta esto último se tendría la siguiente versión:

```
...
si (a > b) entonces
    p ← a
    k ← p+b
    escriba k
si_no
        p ← a
        escriba k
        k ← p+b
fin_si
...
```

La instrucción p ← a es la primera en ejecutarse y se hace se cumpla o no la condición a>b. Esto quiere decir que no hay razón para que la instrucción esté condicionada. Por esta razón, se puede extraer de la estructura de control condicional si de bloque y dejarla inmediatamente antes de ella.

Finalmente, la versión mejorada del diseño inicial sería:

```
...
p ← a
si (a > b) entonces
    k ← p+b
    escriba k
si_no
        escriba k
        k ← p+b
fin_si
...
```

En cuanto a las instrucciones k ← p+b y escriba k hay que tener cuidado porque aunque aparecen en el cuerpo del si y en el del si_no, su orden de ejecución no es el mismo. Obsérvese que en el cuerpo del si se realiza la asignación y luego se escribe el valor de k y en el cuerpo del si_no ocurre lo contrario. Por esta razón, esas dos instrucciones hay que dejarlas en ambas partes de la estructura de control condicional y en ese orden.

Se ha de tener en cuenta que el nuevo diseño debe ser una versión mejorada y equivalente al original en el sentido que en ejecución hagan exactamente lo mismo. También hay que ser cuidadoso en no sacrificar la claridad en aras de la brevedad y menos cuando apenas se está aprendiendo. Este aspecto del diseño es muy importante pero el aprendiz se irá puliendo a medida que vaya ganando experiencia. Entre los estudiantes se tiende generalmente a asociar el buen diseño con la cantidad de instrucciones del diseño lo cual no es cierto necesariamente.

Ejemplo 2. **Mejore** el diseño del siguiente segmento de algoritmo. Justifique el nuevo diseño.

DISEÑO INICIAL	**NUEVO DISEÑO**
... a ← 5 mq (a > 0 y a < n) haga si (a < n y k = p) entonces escriba "Proceso correcto" k←m escriba k si_no si (k ≠ p y a < n)escriba "Proceso errado" escriba k k←m fin_si fin_mq ...	**...** **a ← 5** **mq (a < n) haga** **si (k = p) entonces** **escriba "Proceso correcto"** **k←m** **escriba k** **si_no** **escriba "Proceso errado"** **escriba k** **k←m** **fin_si** **fin_mq** **...**

Justificación. Como la variable a es inicializada en 5 inmediatamente antes de la estructura mq, sobra preguntar si a > 0. Si entra a la estructura mq es porque a < n entonces sobra preguntarlo en la estructura si de bloque. Ahora, a la opción si_no entraría precisamente si k≠ p y, por supuesto, si a < n. Por lo tanto, sobra el condicional que hay dentro de la opción si_no.

EJERCICIOS 5. Mejorando el diseño del código

Mejore el diseño de los siguientes **segmentos** de código y justifiquelo.

5.1

```
...
si (x ≤ y) entonces
        p ← 5
si_no
        si (x = 3)p ← 5
fin_si
...
```