aprenderaprogramar.com

Ejercicio serie suma términos de una sucesión y multiplicación de n impares en pseudocódigo Parte II (CU00218A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel II

Fecha revisión: 2024

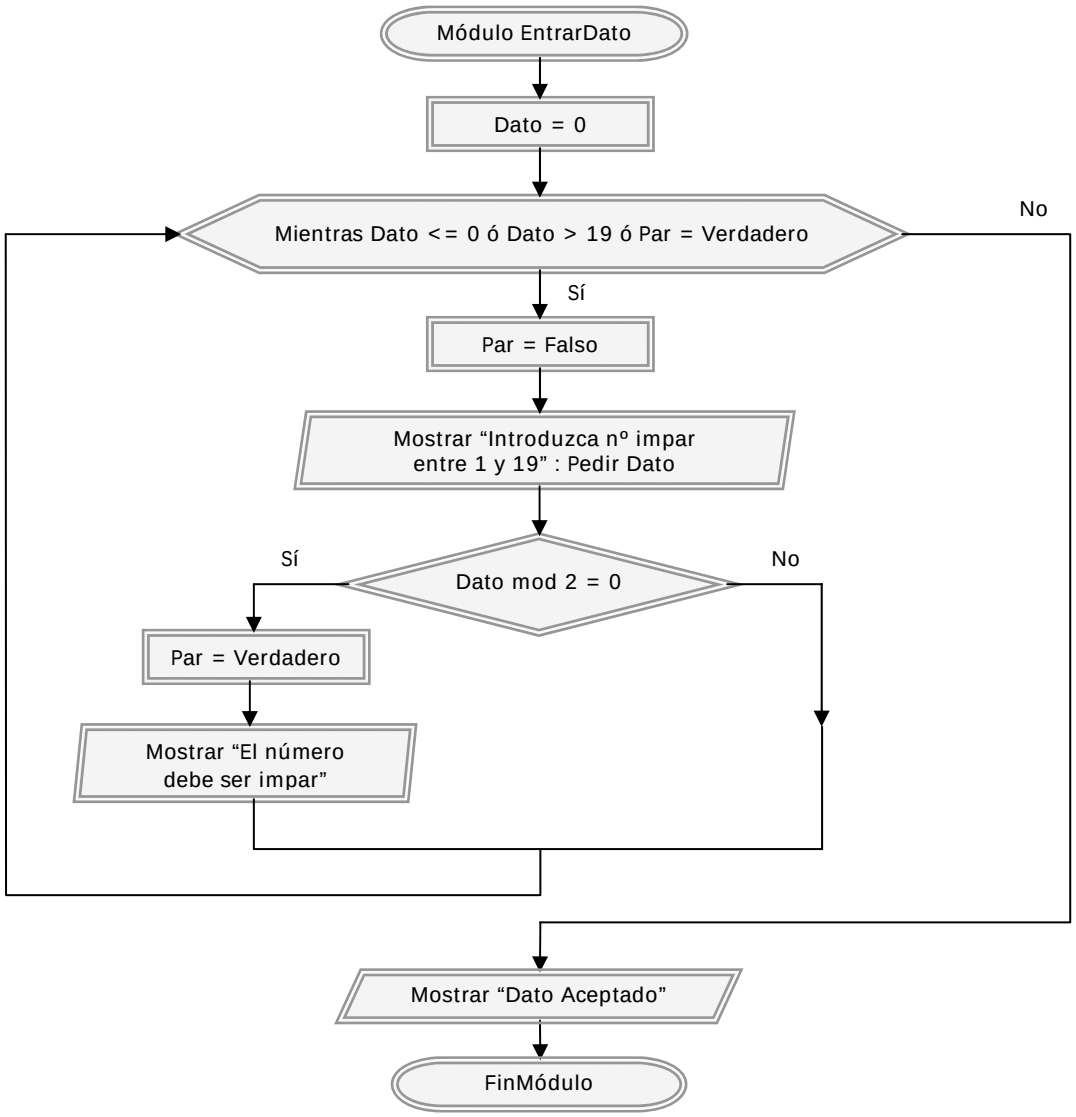
Autor: Mario R. Rancel

Resumen: Entrega nº17 del Curso Bases de la programación Nivel II

24

CONTINUACIÓN DEL EJERCICIO PRESCINDIENDO DEL USO DE ARRAYS

Nota: Proseguimos con los diagramas de flujo y análisis para el programa IMPARES02 que ya hemos planteado en pseudocódigo previamente.



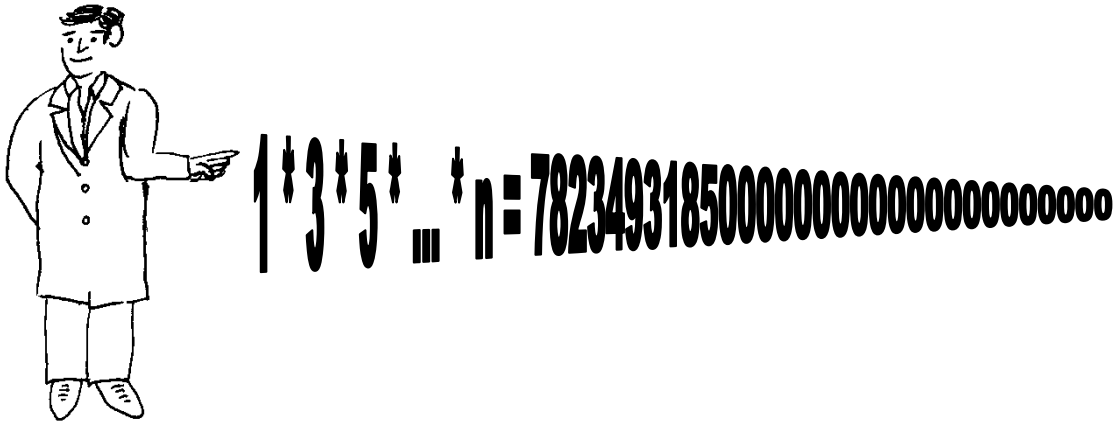
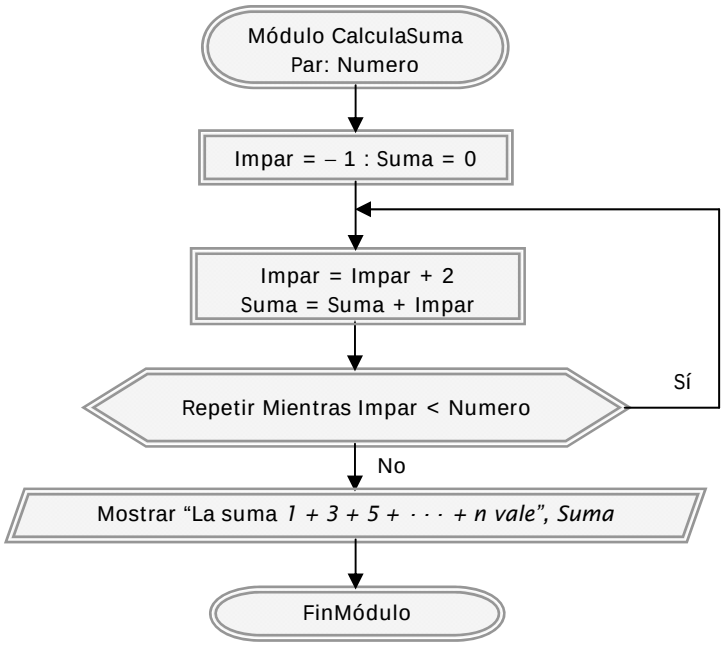
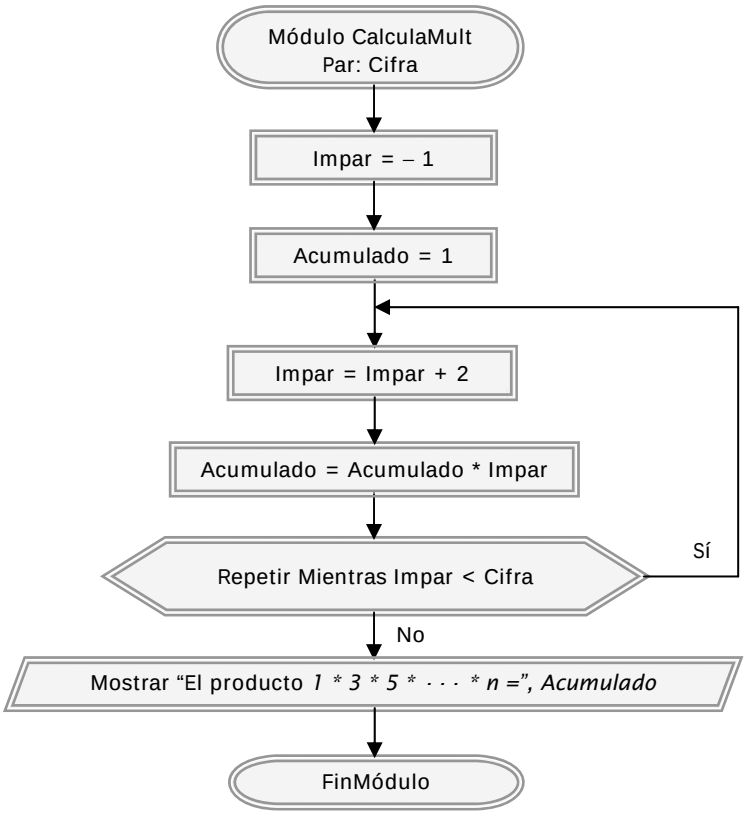


Ilustración: A la hora de programar hemos de atender tanto a los datos de entrada como a los de salida.





Comentarios: El resultado del programa es el mismo que el anterior, aunque en esta ocasión no hemos usado arrays. Cuantos más conocimientos tenemos más vías alternativas existen para llegar a un mismo objetivo. Algunas son más válidas que otras y algunas igual de válidas. En este caso, ¿Mejor usar o no usar arrays? Preferimos no pronunciarnos directamente: depende. En general, será preferible no usar arrays cuando se puede prescindir de ellos, igual que es preferible usar dos variables en vez de cuatro. Pero serán las circunstancias las que manden: si por ejemplo usar 4 variables, pudiendo resolver sólo con 2, introduce una facilidad de lectura y comprensión considerable, puede ser conveniente. Si pensáramos en estos programas como parte de programas más amplios, tendríamos que valorar, que en un caso tenemos almacenada una serie de datos (en el array) y en el otro no. Por tanto, si nos hace falta la serie de datos puede que sea conveniente usar el array.

Hay una cierta ambigüedad en lo que decimos y en esas decisiones nos diferenciamos como programadores. Querer usar siempre arrays o querer usar siempre el mismo tipo de bucles es un error: continuamente hay que estar evaluando la situación para tomar decisiones.

Hasta procesos muy sencillos como determinar si un número es par gozan de vías alternativas.

Veamos lo que serían resultados del programa:

n	*1 + 3 + 5 + · · · + n*	*1 * 3 * 5 * · · · * n*
1	1	1
5	9	15
9	25	945
15	64	2027025
19	100	654729075

A poco que nos fijemos comprobamos que los sumatorios son cuadrados perfectos... ¿Casualidad? Veamos más datos:

3 genera un sumatorio igual a 4
$7 \rightarrow 16$
$11 \rightarrow 36$

Trataremos de hallar una relación:

$n = 3 \rightarrow 2^2$
$n = 5 \rightarrow 3^2$
$n = 7 \rightarrow 4^2$

Expresamos el resultado en función de *n*:

n = 3	→ (n – 1) ²	→	1 = ((3 + 1) / 2) – 1 = ((n + 1) / 2) – 1
n = 5	→ (n – 2) ²	→	2 = ((5 + 1) / 2) – 1 = ((n + 1) / 2) – 1
n = 7	→ (n – 3) ²	→	3 = ((7 + 1) / 2) – 1 = ((n + 1) / 2) – 1
n = 9	→ (n – 4) ²	→	4 = ((9 + 1) / 2) – 1 = ((n + 1) / 2) – 1
n = 11	→ (n – 5) ²	→	5 = ((11 + 1) / 2) – 1 = ((n + 1) / 2) – 1

Llegamos a la conclusión de que para un valor *n* impar el sumatorio propuesto vale:

$$\left[n-\left(\frac{n+1}{2}-1\right)\right]^2$$

Simplificando:

$$\left(n-\frac{n}{2}-\frac{1}{2}+1\right)^2=\left(\frac{n}{2}+\frac{1}{2}\right)^2=\left(\frac{n+1}{2}\right)^2$$

En definitiva:

$$1+3+5+\cdots+n=\left(\frac{n+1}{2}\right)^2$$

Si comprobamos esta fórmula con una calculadora vemos que nos lleva al mismo resultado que teclear todos los números de la serie... Llegamos pues a que existe una vía más eficiente de resolver el sumatorio y que por supuesto nos conviene usar... si caemos en la cuenta. A medida que los programas se hacen complejos es más fácil que aparezcan procedimientos relativamente ineficientes en ellos. Esto tampoco debe obsesionarnos. Ocurre lo mismo que con un proyecto de ingeniería: un programa siempre es mejorable. Y si siempre es mejorable y nos empeñamos en realizar una mejora tras otra, una nueva versión tras otra, nos puede ocurrir que nunca acabemos el programa, o que tardemos demasiado.

El número de mejoras o rectificaciones que será necesario realizar en un programa depende en gran medida de la calidad del material de partida (llámese programa o algoritmo), y ésta depende de la capacidad del programador y especialmente, de que el proceso de construcción del programa sea el correcto. En el ejemplo que nos ocupa, lo importante será desarrollar un algoritmo eficiente que nos resuelva el problema. Si además caemos en la cuenta de que a través de las matemáticas, el uso o no de arrays, un cambio en la organización de variables, etc. podemos realizar notables mejoras, ¡chapeau! Sin perder de vista que el objetivo es solucionar un problema y no necesariamente de la mejor forma posible, si esto supone demasiado tiempo o esfuerzo.

Próxima entrega: CU00219A

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=36&Itemid=60