aprenderaprogramar.com

Estructuras de repetición (bucles). Instrucción desde - siguiente (for - next). Anidamientos. (CU00150A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel I

Fecha revisión: 2024

Autor: Mario R. Rancel

Resumen: Entrega nº 49 del Curso Bases de la programación Nivel I

24

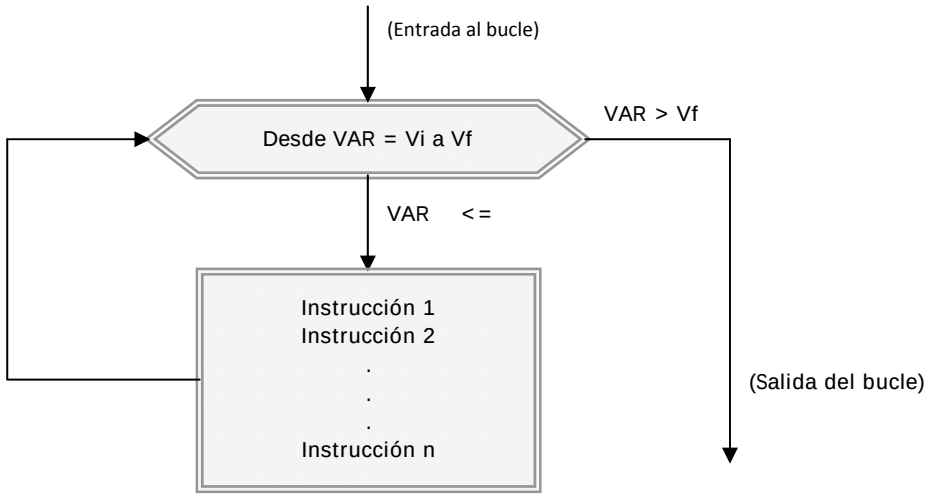
ESTRUCTURAS DE REPETICIÓN (BUCLES). INSTRUCCIÓN DESDE … SIGUIENTE

La instrucción *Desde (Valor Inicial, Valor Final, Paso) ... Siguiente* explota la capacidad del ordenador para repetir procesos y para contar. Muchas veces esta instrucción se puede reemplazar por otras instrucciones o estructuras de tipo repetición. Se podrá optar por la que se estime más oportuna, que muchas veces será la instrucción *Desde ... Siguiente* pues reúne una cualidades interesantes. Escribiremos esta instrucción de la siguiente manera:

```
Desde VAR = Vi hasta Vf Hacer
        Instrucción 1
        Instrucción 2
            .
            .
            .
        Instrucción n
Siguiente VAR
```

El bloque de instrucciones siempre va sangrado respecto a los límites definidos por Desde y Siguiente. *VAR* es una variable de referencia para la instrucción, que toma inicialmente el valor *Vi* (valor inicial). Con la variable de referencia en el valor inicial se procesa el grupo de instrucciones internas hasta llegar a la instrucción *Siguiente*. En este momento el flujo del programa no sigue secuencialmente sino que se vuelve circular, pues se retorna a la instrucción *Desde*, donde *VAR* toma el valor *VAR + 1*. A continuación se vuelve a procesar el conjunto de instrucciones internas y se continua repitiendo el proceso hasta que, tras *n* repeticiones, el valor *VAR* resulta mayor que *Vf* (valor final). Cuando esto se produce, ya no vuelve a haber una repetición del conjunto de instrucciones internas, sino que se continúa la ejecución del programa por la instrucción posterior a *Siguiente*.

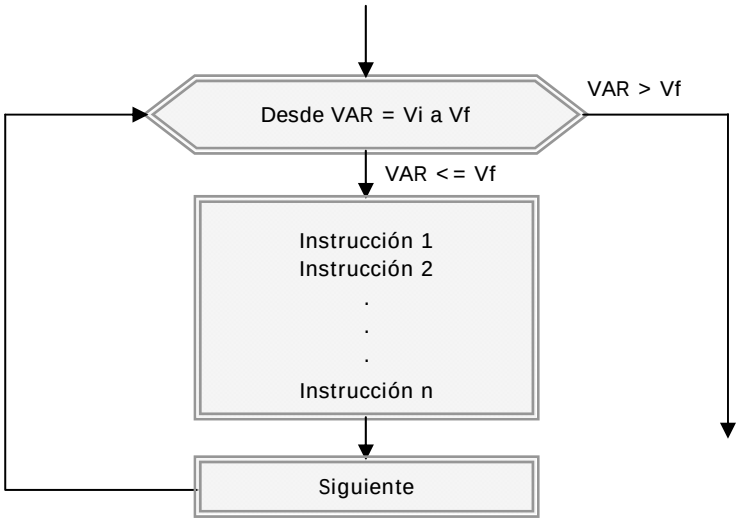
Gráficamente utilizaremos la siguiente representación:



Recordemos, como se expuso para el *Si*, que se consideran válidos para el símbolo de decisión las dos representaciones siguientes:

El símbolo (2) permite un mejor aprovechamiento del espacio cuando el texto a englobar es largo. Para los parámetros de inicio y fin se acepta igualmente *Vi* a *Vf* como *Vi* hasta *Vf* .

La expresión *Siguiente* no tiene representación. El motivo es únicamente la percepción visual y claridad. Al igual que en la instrucción *Si ... Entonces* no representamos la expresión *FinSi* por quedar ésta definida por la fuerza visual de las líneas de flujo, en la instrucción *Desde* prescindimos de representar la expresión *Siguiente* por igual causa. Podríamos emplear un esquema del tipo:



Esta representación es aceptable, pero en un diagrama con decenas de instrucciones *Desde* supondría decenas de símbolos adicionales.

La variable *VAR* suele denominarse “contador” pues es un elemento que toma valores ordenadamente. Así, es habitual utilizar expresiones como “¿Cuál es el valor actual del contador?” o “Voy a iniciar el contador en 6”. El uso de contadores, como veremos más adelante, es de gran utilidad. Entre los programadores es habitual usar letras para identificar a los contadores como *i, j, k, m, n.* No es obligatorio, puesto que al ser variables se les puede asignar cualquier nombre válido para una variable, pero sí creemos que es conveniente al menos como pauta para no cometer errores. Como organización de variables es tan válido:

```
Salario(4) = “C/ Alameda”
i = “José Hdez. Pérez”
Nombre = 1, 2, 3, 4  (contador en un Desde)
Calle3 = 697,32
```

Como:

```
Salario(4) = 697,32
i = 1, 2, 3, 4  (contador en un Desde)
Nombre = "José Hdez. Pérez"
Calle3 = "C/ Alameda"
```

A efectos del ordenador una u otra organización es indiferente. A efectos humanos una mala organización nos puede generar un fuerte dolor de cabeza. Si por cualquier motivo nos encontramos con un programa con variables desorganizadas y hemos de seguir trabajando en él, con seguridad nos ayudará parar y reorganizar las variables antes de perdernos en una maraña incomprensible.

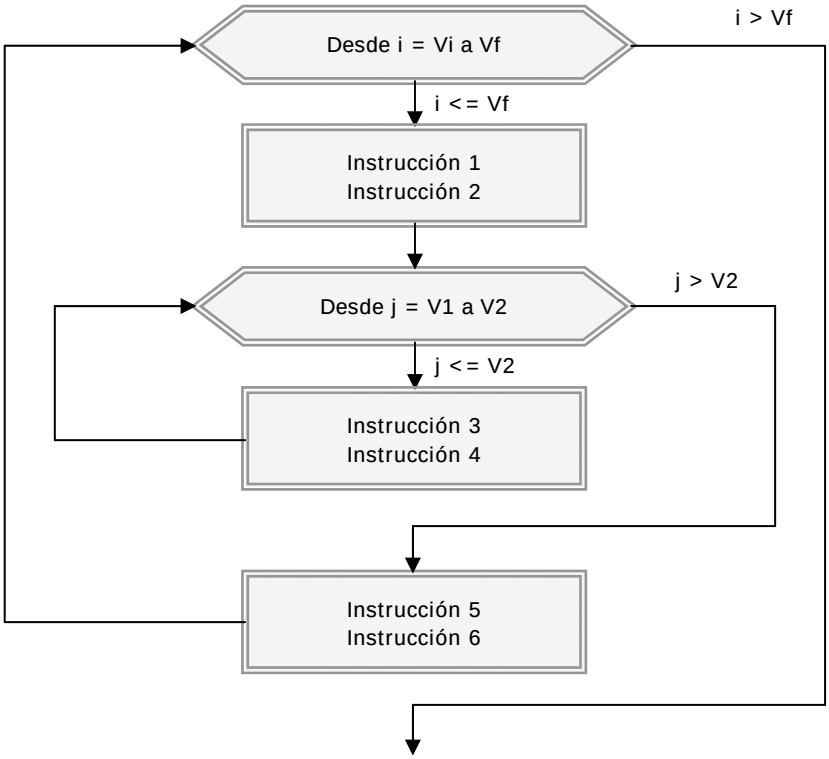
La instrucción *Desde ... Siguiente* es anidable dentro de sí misma mediante la inserción de un bloque *Desde* dentro de otro. Como siempre, recurriremos al sangrado para mantener el pseudocódigo ordenado. Ha de verificarse que todo *Desde* se cierra con un *Siguiente*.

Anidamiento simple (pseudocódigo)

```
[Pseudocódigo aprenderaprogramar.com]
Desde i = Vi hasta Vf Hacer
        Instrucción 1
        Instrucción 2
        Desde j = V1 hasta V2 Hacer
                Instrucción 3
                Instrucción 4
        Siguiente
        Instrucción 5
        Instrucción 6
Siguiente
```

Nota: Si después de siguiente no se especifica variable se entiende que el siguiente hace referencia al *Desde* anterior más próximo.

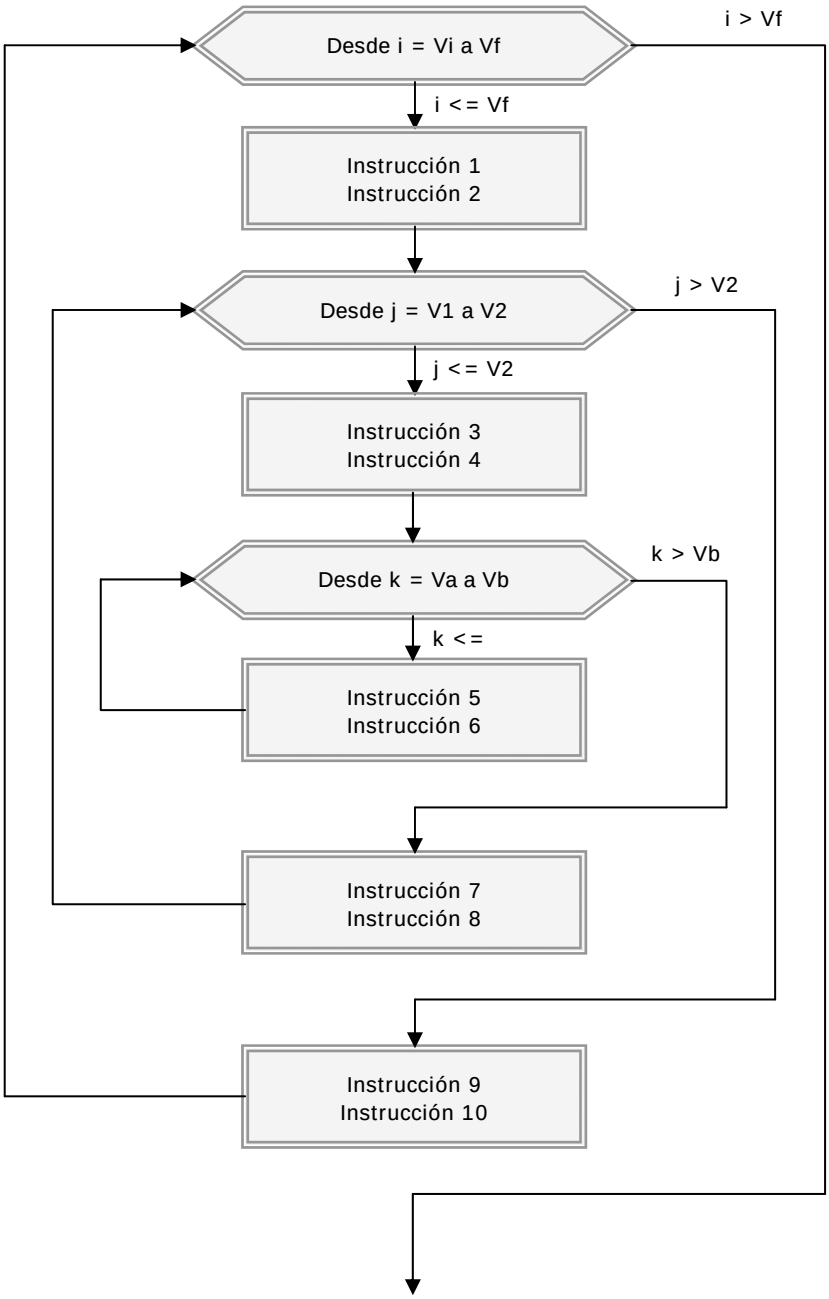
Anidamiento simple (diagrama de flujo aprenderaprogramar.com)



Anidamiento doble (pseudocódigo)

```
[Pseudocódigo aprenderaprogramar.com]
Desde i = Vi hasta Vf Hacer
        Instrucción 1
        Instrucción 2
        Desde j = V1 hasta V2 Hacer
                Instrucción 3
                Instrucción 4
                Desde k = Va hasta Vb Hacer
                        Instrucción 5
                        Instrucción 6
                Siguiente
                Instrucción 7
                Instrucción 8
        Siguiente
        Instrucción 9
        Instrucción 10
Siguiente
```

Anidamiento doble (diagrama de flujo aprenderaprogramar.com)



Los anidamientos se pueden convertir en triples, cuádruples, quíntuples, etc. El principal problema de los anidamientos múltiples será la dificultad para el seguimiento del flujo del programa y confusiones en relación al funcionamiento de los contadores y los valores que adoptan.

Próxima entrega: CU00151A

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=28&Itemid=59