INTRODUCCIÓN

Muchos problemas requieren que ciertas instrucciones se repitan un conjunto finito de veces; es más estas instrucciones utilizarán un conjunto de variables que durante estas repeticiones irán modificando su valor según la lógica que sigan esas instrucciones.

En el mundo de los videojuegos esta situación resulta muy frecuente, por ejemplo puede tener una porción de programa que ejecute un algoritmo que constantemente esté verificando si una bala que nuestro personaje principal ha disparado ha colisionado con un conjunto de enemigos presentes en la pantalla; o por ejemplo, suponga que desarrolla un juego en 2D y debe dibujar el interior de un castillo para lo cual solo posee imágenes de bloques (como en Super Mario Bros), Ud. podría usar estas estructuras repetitivas para indicar que esos bloques conformen el interior de la escena sin tener que explicitar una instrucción por cada bloque usado.

En esta sección se examinarán los diferentes métodos que utilizan los diseñadores de juegos para construir secciones de instrucciones repetitivas. Se describe y analiza el concepto de bucle, iteración, bucle infinito y las diferentes estructuras de control iterativas.

CONCEPTOS

- Estructura de control iterativa: es un tipo de estructura de control que permite repetir una o varias acciones (instrucciones o sentencias) un determinado número de veces.
- Bucle: se indica con este nombre a la sección de código que se repite. Es decir, dentro de una estructura de control iterativa hay un bucle, por el cual luego de ejecutar su última instrucción saltará a la primera de cumplirse cierta condición.
- Iteración: Cada repetición de un bucle se conoce como iteración.

FUNCIONAMIENTO GENERAL DE LAS ESTRUCTURAS DE CONTROL ITERATIVAS

Las dos principales preguntas para realizarse en el diseño de un bucle son:

1. ¿qué contiene el bucle? Y
2. ¿cuántas veces se debe repetir?

La primera pregunta tiene por fin cuestionarse en primer lugar si es necesario utilizar una estructura de control iterativa y en segundo lugar analizar y entender que es lo que se debe repetir.

La segunda pregunta lleva a la afirmación de que es necesario saber cuántas iteraciones se esperan que se realicen. Es decir, no plantea indicar el número exacto de iteraciones, pero si tener la seguridad de que ese número es finito.

Esto nos lleva a plantearnos un aspecto esencial: debe existir un mecanismo que provoque que en algún momento las iteraciones se detendrán, caso contrario jamás se saldría de la estructura de control y se produciría lo que se conoce con el nombre de **bucle infinito**.

Veamos este planteo en un ejemplo representativo; sea la siguiente operación matemática

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Como puede observar es el cálculo del factorial de 5 (o también denominado 5 factorial), que por definición implica multiplicar todos los números enteros en el rango de [1, 5].

Observe que aquí hay una operación (que es el producto) que se podría repetir (respuesta a la primera pregunta) 5 veces (respuesta a la segunda pregunta) con el objetivo de obtener el valor de esta función. Puede notar que existe una operación a realizar un número finito de veces y que existe una condición de parada para las iteraciones.

Se debe recalcar que existen ocasiones en que la única manera de resolver un problema es mediante estructuras iterativas.

Volviendo al caso anterior, ¿Cómo lo resolvería sin estructuras iterativas?

NOMBRE DE LA ENTIDAD QUE RESUELVE EL PROBLEMA: Factorial
VARIABLES valorFactorial: Entero
NOMBRE DEL ALGORITMO: obtener_factorial **PROCESO DEL ALGORITMO** 1. valorFactorial ←5*4*3*2*1 2. *Mostrar* valorFactorial

Como puede verificar, este caso de algoritmo satisface el objetivo para el que ha sido creado. El Mostrar siempre muestra el resultado de una variable.

Ahora vayamos más allá, con el objetivo de que el algoritmo tenga una mayor utilidad suponga que deseamos que el usuario ingrese un valor entero y se obtenga el factorial de este.

Si intenta diseñar este algoritmo sin usar estructuras iterativas o funciones recursivas (concepto se estudiará más adelante) es imposible lograrlo.

Para situaciones como esta es que vamos a necesitar conocer y aplicar las estructuras iterativas que compartiremos a continuación.

ESTRUCTURA ITERATIVA MIENTRAS

La estructura iterativa **`mientras`** (en inglés while) es aquella en la que el bucle se repite mientras se cumple una determinada condición. Cuando se ejecuta la instrucción **`mientras`**, la primera cosa que sucede es que se evalúa la condición (una expresión lógica o booleana).

Si el resultado de la evaluación devuelve **falso**, no se ejecutará el bucle continuando la ejecución del algoritmo fuera de la estructura **`mientras`** (se dice que salta hacia afuera del **`fin_mientras`**). Si la expresión booleana retorna **verdadero**, entonces se ejecuta el bucle y al llegar al final de este se procederá a evaluar nuevamente la expresión booleana.

Por lo tanto, las iteraciones se sucederán una tras otra mientras la expresión booleana (condición) siga retornando el valor **verdadero**. A continuación, se esquematiza la estructura de control iterativa **`mientras`**

Pseudocódigo en español

```
mientras condicion hacer
   accion S1
   accion S2
   .
   .
   acción Sn
fin _ mientras
```

Pseudocódigo en inglés

```
while condicion do
   <acciones>
   .
   .
endwhile
```

Como conclusión podemos decir:

1) En esta estructura iterativa **primero se evalúa la condición**.
2) Si no se verifica la condición no se ejecutarán las instrucciones del bucle
3) Se hace necesario que una porción del bucle opere de tal manera que en algún momento se provoque que la evaluación de condición devuelva falso para salir del proceso iterativo, caso contrario entraremos en un bucle infinito y el algoritmo jamás finalizará.

Ejercicio: Resolver el problema del cálculo del factorial usando la estructura iterativa **`mientras.`**

Análisis del Problema

- **Especificación el Problema**: Diseñar un algoritmo que muestre el factorial de cualquier número entero positivo que ingresa el usuario.
- **Análisis**:
  Datos de Entrada:
  numIngresado: Entero //un entero que representa el número ingresado

  Datos de salida:

  n:Entero // el valor del factorial de numIngresado

  Proceso:
  ¿Quién realiza el proceso?: Factorial
  ¿Cuál es el proceso que se realiza?
  La variable del que se va a obtener el factorial matemáticamente se denomina **n**. Sabemos que el factorial de un número se define de manera simbólica como

$$n! = 1 \times 2 \times 3 \times 4 \times \ldots \times (n-1) \times n.$$

  Lo cual es expresado matemáticamente como

$$n! = \prod_{k=1}^{m} k$$

  Y además $0! = 1$

  Lo anterior representa una repetición finita de multiplicaciones. Esto se ajusta a la estructura iterativa **mientras** donde **m** es determinada por el valor de numIngresado.

Diseño del Algoritmo

NOMBRE DE LA ENTIDAD QUE RESUELVE EL PROBLEMA: Factorial
VARIABLES numIngresado: Entero n: Entero // el valor del factorial de numIngresado k: Entero // permite llevar la cuenta de multiplicaciones
NOMBRE DEL ALGORITMO: obtener_factorial **PROCESO DEL ALGORITMO** 1. *Leer* numIngresado 2. k←1 3. n←1 4. **mientras** (k <= numIngresado) **hacer** 5. n ←n*k 6. k++ **7. fin_mientras** 8. *Mostrar* n

Observe que el elemento clave para que en algún momento la ejecución del algoritmo salga de la estructura está en la alteración del valor de la variable **k** en cada bucle. Esto se logra cuando **k** no es menor o igual que `numIngresado`, por lo tanto, la evaluación de la condición devuelve falso y en consecuencia no ingresará al bucle.

UNA VARIABLE CON OBJETIVO ESPECÍFICO: EL CONTADOR

Se define como **variable contador** a aquella cuyo objetivo es contar la ocurrencia de ciertos sucesos; para lo cual cada vez que se presenta esta ocurrencia hay que aumentar el valor de la variable en uno, lo cual se logra de la siguiente manera:

a) Suponga que posee una variable cuyo nombre es **`contador`**
b) En algún momento se inicia esa variable con el valor 0 (cero) es decir

contador ← 0

c) Al momento de la ocurrencia del suceso a contar se debe realizar lo siguiente

contador ← contador + 1

Observe que en el mundo de los algoritmos es posible sumar al valor de una variable un valor y asignárselo a la misma variable.
Por otro lado, la expresión anterior se puede escribir usando operadores unitarios, de la siguiente manera

contador++ o ++contador

Ud puede observar que en el ejercicio del factorial, k hace de contador.

ESTRUCTURA ITERATIVA HACER-MIENTRAS

El bucle **`mientras`** evalúa la expresión al comienzo del bucle de repetición; siempre se utilizan para crear bucle **pre-test**. Los bucles pre-test se denominan también bucles controlados por la entrada. En numerosas ocasiones se necesita que el conjunto de sentencias que componen el cuerpo del bucle se ejecute al menos una vez sea cual sea el valor de la expresión o condición de evaluación.

Estos bucles se denominan bucles **post-test** o bucles controlados por la salida. Un caso típico es el bucle **`hacer-mientras`** (do-while).

El bucle **`hacer-mientras`** es análogo al bucle **`mientras`** y el cuerpo del bucle se ejecuta una y otra vez mientras la condición (expresión booleana) sea verdadera. Existe, sin embargo, una gran diferencia y es que el cuerpo del bucle está encerrado entre las palabras reservadas **`hacer`** y **`mientras`**, de modo que las sentencias de dicho cuerpo se ejecutan, al menos una vez, antes de que se evalúe la expresión booleana. En otras palabras, el cuerpo del bucle siempre se ejecuta, al menos una vez, incluso aunque la expresión booleana sea falsa.

A continuación, se esquematiza la estructura de control iterativa **`mientras`**

```
hacer
   <acciones>
mientras  (<expresión>)
```

```
dowhile  condicion
   <acciones>
   .
   .
enddo
```

Ejercicio: Realice un programa que lea un número (por ejemplo 198) y que obtenga el número inverso (por ejemplo 891) **.**

Análisis del Problema

- **Especificación del Problema**: Dado un número entero invertirlo y mostrarlo por pantalla.
- **Análisis**:
  Datos de Entrada:
  numIngresado: Entero //un entero que representa el número ingresado

  Datos de salida:

  digitoDelNumInvertido:Entero // cada uno de los dígitos del número invertido

  Proceso:
  ¿Quién realiza el proceso?: Persona que realiza el cálculo
  ¿Cuál es el proceso que se realiza?

  Es posible obtener los valores de cada dígito del número inverso al sucesivamente obtener el módulo del número ingresado cuando se lo divide por 10 y posteriormente hacer que el número ingresado se divida por 10.

  Estos pasos lo repetiremos siempre que el número ingresado sea mayor que cero (0).

Diseño del Algoritmo

NOMBRE DE LA ENTIDAD QUE RESUELVE EL PROBLEMA: Persona Calculando
VARIABLES numIngresado: Entero digitoDelNumInvertido: Entero
NOMBRE DEL ALGORITMO: obtener_numero_invertido **PROCESO DEL ALGORITMO** 1. *Leer* numIngresado 2. *Mostrar* numIngresado 3. **hacer** 4. digitoDelNumInvertido ←(numIngresado % 10) 5. *Mostrar* digitoDeNumInvertido 6. numeroIngresado ← numeroIngresado / 10 **7. mientras** (numIngresado > 0)

Con cada iteración se obtiene el dígito más a la derecha, ya que es el resto de la división entera del valor del número (**`numIngresado`**) por 10.

Así, suponiendo que el valor que ingresa el usuario es 198, entonces en la primera iteración **`digitoDelNumInvertido`** vale 8 ya que es el resto de la división entera de 198 entre 10 (cociente 19 y resto 8), correspondiente a la línea 4.

En la línea 5 se visualiza el valor 8.

A continuación (línea 6) se divide 198 entre 10 y se toma el cociente entero 19, que se asigna a la variable **`numeroIngresado`**.

En la siguiente iteración se divide 19 por 10 (cociente entero 1, resto 9) y se visualiza, por consiguiente, el valor del resto, **`digitoDelNumInvertido`**, es decir el dígito 9. A continuación se divide 19 por 10 y se toma el cociente entero, es decir, 1.

En la tercera y última iteración se divide 1 por 10 y se toma el resto (**`digitoDelNumInvertido`**) que es el dígito 1.

Se visualiza el dígito 1 a continuación de 89 y como resultado final aparece 891. A continuación, se efectúa la división de nuevo por 10 y entonces el cociente entero es 0 que se asigna a **`numeroIngresado`** que al no ser ya mayor que cero hace que se termine el bucle y el algoritmo correspondiente.

ESTRUCTURA ITERATIVA PARA

Esta estructura posee el siguiente esquema

```
para v ← vi hasta vf [incremento incr] hacer
  <acciones>
     .
     .
     .
fin_para
```

La estructura **`para`** comienza con un valor inicial (**`vi`**)de la variable **`v`** que actúa como índice y las acciones especificadas se ejecutan, a menos que el valor inicial sea mayor que el valor final, que se representa por **`vf`**.

La variable índice (**`vi`**) se incrementa en uno por defecto, o un valor mayor según lo expresado por **`incr`** y si este nuevo valor no excede al final (**`vf`**), se ejecutan de nuevo las acciones.

Por consiguiente, las acciones específicas en el bucle se ejecutan para cada valor de la variable índice desde el valor inicial hasta el valor final con el incremento indicado en **`incr`**.

El incremento de la variable índice siempre es 1 si no se indica expresamente lo contrario. Dependiendo del tipo de lenguaje, es posible que el incremento sea distinto de uno, positivo o negativo.

La variable índice o de control normalmente será de tipo entero y es normal emplear como nombres las letras i, j, k.

El formato de la estructura desde varía si se desea un incremento distinto a 1, bien positivo, bien negativo (decremento).

Ejemplo: Modifique el diseño del algoritmo que obtiene el factorial de un número usando la estructura iterativa **`para`**.

NOMBRE DE LA ENTIDAD QUE RESUELVE EL PROBLEMA: Factorial
VARIABLES numIngresado: Entero n: Entero // el valor del factorial de numIngresado k: Entero // permite llevar la cuenta de iteraciones
NOMBRE DEL ALGORITMO: obtener_factorial **PROCESO DEL ALGORITMO** 1. *Leer* numIngresado 2. n←1 3. **para** k ← 1 **hasta** numIngresado **incremento** 1 **hacer** 4. n ←n*k **5. fin_para** 6. *Mostrar* n

BIBLIOGRAFIA

Fundamentos de Programación: Algoritmos, estructuras de datos y objetos. Cuarta edición. Luis Goyanes Aguilar. ISBN: 978-84-481-6111-8