Instrucciones para la realización de iteraciones

Muchos problemas del mundo real requieren ser resueltos por medio de procesos repetitivos, es decir, realizar una y otra vez una serie de pasos hasta haber acabado un producto o tarea. Por ejemplo:

a) Excavar un agujero de cierta profundidad: habrá que palear y medir repetidamente hasta alcanzar la profundidad deseada.

b) Levantar una pared de determinada altura: se colocará una hilera de ladrillos, luego otra y otra, hasta llegar a la altura requerida.

c) Cualquiera de las etapas de un proceso de fabricación de un producto en una fábrica, por ejemplo, colocar el tapón en una botella. Se coloca el tapón en una y en otro y en otra, ...

d) Así también la aplicación de todos los pasos del proceso en la fabricación de cada producto es repetitivo.

e) Tomar las muestras de sangre a un paciente hospitalizado. Todos los días por las mañanas, "sin falta", las enfermeras pasan pinchando a los pacientes para que se realicen los exámenes de Laboratorio correspondientes y ver el resultado de la evolución del paciente.

En el área de las matemáticas también hay procesos que se repiten para realizar ciertos cálculos. Por ejemplo:

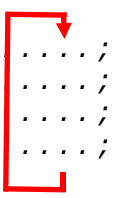
a) La serie de Fibonacci: a partir de dos valores iniciales, del tercero en adelante se calculan sumando los dos términos que le preceden.

b) El factorial de un número: requiere de la realización repetida de los productos de cada entero positivo menor o igual al número, exceptuando el cero.

c) El proceso de cálculo de una integral: la teoría nos dice que es la suma de muchas pequeñas áreas bajo la curva de la función a integrar, entre dos valores de interés.

Siendo así las cosas, nuestro repertorio de instrucciones de C/C++ -y de cualquier otro lenguaje de programación- no estaría completo si no nos permitiera ejecutar un bloque de instrucciones de manera repetida. Por fortuna contamos con una serie de instrucciones que nos permiten realizar lo que en el lenguaje informático se conoce con varios sinónimos: repetición, iteración, bucle, ciclo o lazo. Todos estos términos dan la idea de que algo puede regresar sobre sí mismo.

En forma gráfica, el funcionamiento de una instrucción que itera sobre un bloque de instrucciones de programa, se muestra a continuación.



En este esquema lo que se plantea es que, una vez llegado al final del bloque, hay un salto hacia arriba, al inicio del mismo, y se vuelven a ejecutar las instrucciones del bloque nuevamente. Se resalta la idea de la ejecución no secuencial porque de un punto del programa se salta a otro por el que ya se había pasado antes, en lugar de seguir hacia adelante.

Al igual que con las instrucciones para la toma de decisiones, la vuelta a ejecutar el mismo bloque de debería depender del estado de ciertas variables, mismas que se pueden controlar dentro del bloque de instrucciones.

Clasificación de las instrucciones iterativas

Entre las estructuras de control de flujo que permiten realizar repetición, o iteración, de bloques de instrucciones, están:

a) Las que permiten repetir un conjunto de instrucciones cierta cantidad de veces, conocidas como instrucciones de iteración por conteo.

b) Las que permiten repetir un conjunto de instrucciones de manera condicionada, conocidas como instrucciones de iteración por condición.

Las veremos a continuación.

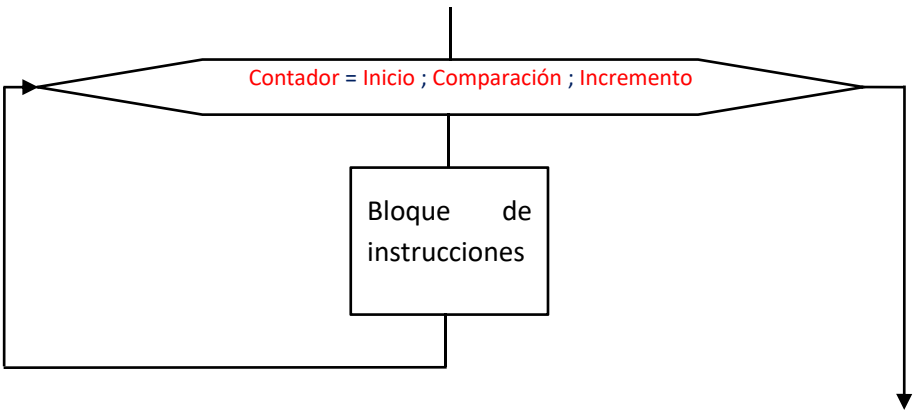
Iteraciones por conteo

Existe una instrucción de este tipo en C/C++, que es la instrucción *for*.

Instrucción *for*

Esta instrucción realiza el conteo de las repeticiones del conjunto de instrucciones sobe el que itera, con la ayuda de una variable que se conoce como contador. La cantidad de veces que se repetirá el bloque suele definirse antes de ejecutarlo la primera vez y el contador va contando las iteraciones que se realizan, hasta alcanzar la cantidad de veces que se desea repetir.

El diagrama de flujo de la instrucción es:



Donde:

- Contador es una variable que se utiliza como contador de la cantidad de iteraciones. Estas iteraciones se definen dentro de un intervalo de valores dentro del cual se realizará el conteo.

- Inicio es el valor inicial que asume la variable contador. Si para efectos prácticos nos imaginarnos un intervalo de conteo, podemos llamarle limite1.

- Comparación es la expresión con la que se verifica si el contador ya finalizó el conteo. En general el contador se compararía con el otro extremo del intervalo, al que podemos llamar limite2. Cuando esta comparación deje de cumplirse, entonces la iteración finalizará.

- Incremento es la expresión que le suma al contador el "tamaño de paso". Este tamaño de paso generalmente se asume constante y puede ser positivo o negativo.

El incremento puede ser positivo para realizar un conteo ascendente, o negativo para realizar un conteo descendente. Pueden suceder las siguientes situaciones:

Relación entre los límites	**Signo del incremento**	**Observación**
limite1 <= limite2	+	Se realizan iteraciones. El conteo es ascendente.
limite1 <= limite2	-	Se provoca un "lazo infinito". Debe tenerse cuidado.
limite1 >= limite2	-	Se realizan iteraciones. El conteo es descendente.
limite1 >= limite2	+	Se provoca un "lazo infinito". Debe tenerse cuidado.

La segunda y cuarta entradas de la tabla señalan que se produce lo que se llama un "lazo infinito". Esto quiere decir que la iteración nunca termina. O al menos terminará si hay un error por rebosamiento en le contador, es decir, que se sobrepasa su capacidad de almacenamiento. Pero, en este caso, el programa fallaría en tiempo de ejecución. Este lazo infinito se provoca porque el contador, en lugar de avanzar hacia el limite2, avanza en sentido contrario y, por lo tanto, nunca lo alcanzará ni lo sobrepasará. Así que la condición que termina el lazo nunca se cumplirá.

Esto no quiere decir que, en ocasiones, no se elaboren intencionadamente lazos infinitos. Habrá situaciones en las que sí deseemos implementar un lazo infinito, pero la mayoría de las veces se da por un error de lógica o descuido a la hora de escribir el programa.

Dentro de un programa, un conteo ascendente típico, de uno en uno, con la instrucción *for* se ve así:

```
for(contador = limite1 ; contador <= limite2 ; contador++)
{
   . . . . . ;
```

```
    . . . . . ;
    . . . . . ;
    . . . . . ;
}
```

Aunque debe tenerse en cuenta que el signo de comparación, los límites y el incremento pueden variar de acuerdo a la necesidad del problema.

C/C++ puede identificar el bloque de instrucciones sobre el cuál hay que iterar, debido a que está a continuación de la palabras reservada *for* y encerrado entre llaves. Recuerde que las llaves pueden omitirse si el bloque consta de una sola instrucción.

Ejemplos:

1) Escriba un lazo *for* para iterar de 1 a 6. En cada iteración despliegue el contador para así verificar que este cambia de valor en cada repetición del bloque de instrucciones.

```
#include "iostream"

using namespace std;

int main(void)
{
      int i;

      cout << "CONTAR DE 1 A 6" << endl << endl;

      for (i = 1; i <= 6 ; i = i + 1)
      {
            cout << i << endl;
      }

      cout << endl;
}
```

En este ejemplo el bloque de instrucciones sobre el que se itera es únicamente una instrucción., así que las llaves pueden ser omitidas. Esta instrucción despliega el valor de *i*, la cual es la variable que se utiliza como contador del lazo, vea el encabezado de la instrucción. La variable *i* cambia automáticamente de valor luego de cada iteración, así que a ella la utilizamos para desplegar el conteo en pantalla.

2) Elabore un programa que imprima su nombre cinco veces.

```
#include "iostream"

using namespace std;

int main(void)
{
      string nombre;
      int i;

      cout << endl;
```

```
    cout << "IMPRIMIR SU NOMBRE CINCO VECES" << endl << endl;

    cout << "Digite su nombre: ";
    cin >> nombre;


    for (i = 1; i <= 5; i = i + 1)
        cout << nombre << endl;

    cout << endl;
    return 0;
}
```

En este ejemplo no se despliega el contador. Lo que se despliega es el nombre digitado desde teclado, la cantidad de veces que el contador determina que el lazo debe realizarse Note también que se le han omitido las llaves al bloque de la instrucción *for* porque solo se itera sobre una instrucción.

3) Modifique el programa del problema anterior para que, además de su nombre, despliegue el valor del contador en cada iteración y que el conteo se haga de forma descendente. Además, su nombre debe imprimirse la cantidad de veces que se solicite desde teclado.

```
#include "iostream"

using namespace std;

int main(void)
{
    string nombre;
    int i, n;

    cout << endl;
    cout << "IMPRIMIR SU NOMBRE Y CONTEO DESCENDENTE" << endl << endl;

    cout << "Digite su nombre: ";
    cin >> nombre;
    cout << "¿Cuantas veces desea imprimirlo? ";
    cin >> n;

    for (i = n; i >= 1; i = i - 1)
    {
        cout << i << "      ";
        cout << nombre << endl;
    }

    cout << endl;
    return 0;
}
```

4) Sumar todos los enteros que se encuentran en un intervalo [a, b]. Solicite los extremos del intervalo desde teclado y asegúrese que *a* sea menor o a lo sumo igual a *b*.

```
#include "iostream"

using namespace std;
```

```
int main(void)
{
	int a, b, i, suma;

	cout << endl;
	cout << "SUMAR LOS ENTROS CONTENIDOS EN UN INTERVALO" << endl << endl;

	cout << "Digite el límite inferior del intervalo: ";
	cin >> a;
	cout << "Digite el límite superior del intervalo: ";
	cin >> b;

	if(a > b)
		cout << "Debe digitar los límites al revés" << endl;
	else
	{
		suma = 0;
		for (i = a; i <= b; i = i + 1)
			suma = suma + i;
	}

	cout << "La suma de los enteros del intervalo es: " << suma << endl;

	cout << endl;
	return 0;
}
```

Iteraciones por condición

La instrucción por conteo vista arriba puede ser utilizada cuando se conoce la cantidad de iteraciones que se deben realizar, o cuando los límites del intervalo de iteración pueden ser, de alguna manera, calculados o proporcionados. Pero hay cierto tipo de problemas que deben resolverse por repetición de la tarea, pero no se puede llegar a establecer de antemano la cantidad de iteraciones a realizar. En este tipo de problemas es típico que, en el mismo proceso repetitivo se vaya determinando si ya es momento de terminar o aun se debe continuar iterando.

En los casos en los que no se conoce la cantidad de repeticiones, pero sí se conoce el estado o situación a la que se pretende llegar, es bueno valerse de una condición para determinar si se itera o no. Ese estado, meta o situación a la que se quiere llegar lo podemos describir por medio de, al menos, una variable, cuyo valor se va a estar revisando en cada iteración para verificar si el estado deseado ya se ha alcanzado.

Para ello C/C++ cuenta con dos instrucciones:

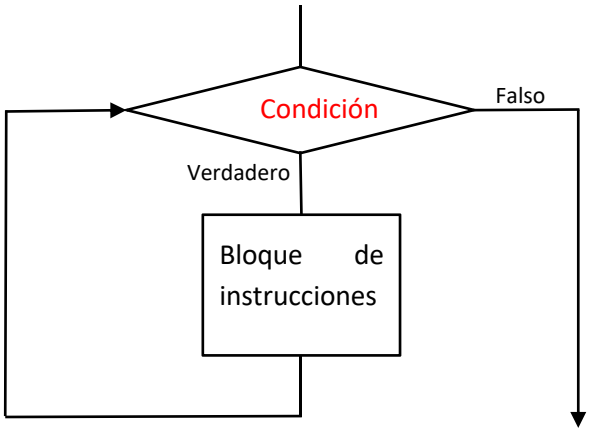
a) La instrucción *while(condición)* ....

b) La instrucción *do* ... *while(condición)*.

Estas instrucciones se describen a continuación.

instrucción *while(condición) ...*

El diagrama de flujo de esta instrucción es:



Donde:

- Puede notarse que hay una Condición antes del bloque de instrucciones, la cual determina, mediante su valor de verdad, si el bloque se ejecuta, o no.

- Si la Condición es verdadera, el bloque se ejecuta. Pero si es falsa, notar que la línea de flujo de la derecha conduce hacia afuera de la instrucción. Llevando la ejecución hacia la siguiente instrucción del programa.

Dado que la condición está antes del bloque de instrucciones, si al evaluarla la primera vez resulta ser falsa, el bloque no se ejecutará ninguna vez.

Dentro del programa, la instrucción *while* se ve así:

```
while(Condición)
{
   . . . . . . . . ;
   . . . . . . . . ;
   . . . . . . . . ;
   . . . . . . . . ;
   . . . . . . . . ;
}
```

C/C++ puede identificar el bloque de instrucciones sobre el cuál hay que iterar, debido a que está inmediatamente después de la palabra reservada *while* y está contenido entre llaves. Obviamente las llaves pueden omitirse si el bloque consta de una sola instrucción.

IMPORTANTE: En general, todos los programas que se pueden resolver con un lazo por conteo, se pueden resolver con lazos por condición. Pero no todos los problemas que se pueden resolver por medio de lazos por condición podrán resolverse con lazos por conteo. Así que todos los problemas resueltos en el tema anterior se pueden volver a resolver en este tema, haciendo las consideraciones para el cambio de instrucción.

Si se desea resolver un problema que implica cierto número de iteraciones conocido, utilizando un lazo while, la estructura típica es la siguiente:

```
contador = limite1;
while(cantador <= limite2)
{
    ...
    Instrucciones a realizar;
    ...
    contador = contador + incremento;
}
```

Note que todos los requisitos para plantear correctamente una instrucción *for*, están colocados como instrucciones de programa al utilizar una sentencia *while*, ya que esta solo realiza la comparación que tiene a su derecha para determinar si continúa o no iterando.

Ejemplos:

1) Realice un conteo de uno a 6 utilizando la instrucción *while*. Luego compare el programa con la versión de conteo utilizando *for* que se hizo arriba.

```
#include "iostream"

using namespace std;

int main(void)
{
    int i;

    cout << "CONTAR DE 1 A 6" << endl << endl;

    i = 1;
    while(i <= 6)
    {
        cout << i << endl;
        i = i+ 1;
    }

    cout << endl;
}
```

Note que al contador se le asigna el valor inicial en una línea del programa, el incremento del contador también se debe realizar en una línea del programa y la comparación con el límite superior se especifica en el encabezado de la instrucción. En otras palabras, si una iteración no puede ser establecida con *for*, C/C++ le permite al programador establecer todo lo que es necesario realizar, por medio de una instrucción *while*.

2) La conjetura de Collatz (o conjetura de Ullman) establece que, para cualquier número natural, a través de la aplicación repetida de estas operaciones:

$$f(n)=\begin{cases} n/2, si \quad n \quad es \quad par \\ 3*n+1, si \quad n \quad es \quad impar \end{cases}$$

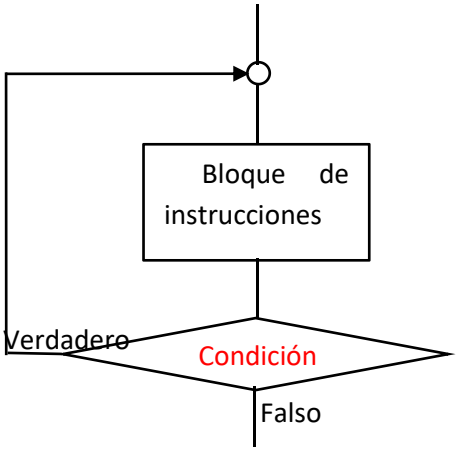
Siempre se obtendrá un uno. Escriba un programa que, dado un número entero positivo, aplique las operaciones de Collatz hasta llegar a uno. Muestre todos los valores obtenidos.

```
#include "iostream"

using namespace std;

int main(void)
{
	int n;

	cout <<"APLICACIÓN DE LA CONJETURA DE COLLATZ\n\n";
	cout << "Digite un número entero positivo: ";
	cin >> n;

	cout << n << "    ";
	while(n > 1){
		if(n % 2 == 0)
			n = n / 2;
		else
			n = 3 * n + 1;
		cout << n << "    ";
	}

	cout << endl;
	return 0;

}
```

En este problema se sabe que el proceso iterativo implica la aplicación de alguna de las dos fórmulas que se han definido, pero no se sabe cuantas veces habrá que realizar la operación. Lo único que se sabe es que en algún momento *n* llegará a ser *1*. Debido a ello es que no podemos implementar una iteración por conteo, sino que debemos implementarla por condición. La condición para mantenerse iterando es, precisamente, que el valor siga siendo mayor que *1*.
Notar que el bloque de instrucciones dentro del lazo consiste de dos instrucciones. La primera es una instrucción *if* de dos bloques. Esto se debe a que, en cada iteración, se aplicará una u otra fórmula de acuerdo a si *n* es par o impar. La segunda instrucción es un *cout*, pues el valor de *n* será desplegado después de haber sido recalculado.

instrucción *do ... while(condición)*

El diagrama de flujo de esta instrucción es:



Donde:

- Puede notarse que hay una Condición después del bloque de instrucciones, la cual determina, mediante su valor de verdad, si el bloque se ejecuta o no, nuevamente.

- Si la condición es verdadera, el bloque se ejecuta. Pero si es falsa, el proceso iterativo termina, llevando la ejecución hacia la siguiente instrucción del programa.

Dado que la condición está después del bloque de instrucciones, éste deberá ejecutarse una vez antes de la primera evaluación de la condición. Por tanto, al implementar esta instrucción, el bloque de instrucciones se realiza, como mínimo, una vez.

La sintaxis de esta instrucción es:

```
do
{
   . . . . . ;
   . . . . . ;
   . . . . . ;
}
while(Condición);
```

C/C++ puede identificar el bloque de instrucciones sobre el cuál hay que iterar, debido a que está entre las palabras reservadas *do* y *while*. Si el bloque consta de más de una instrucción debe estar encerrado entre llaves.

A menudo un mismo problema puede ser resuelto aplicando cualquiera de las dos instrucciones de iteración por condición. La diferencia entre usar una u otra dependerá de la facilidad que ambas presenten para resolver un problema concreto o de la preferencia que tenga el programador.

Ejemplos:

1) Realice, una vez más, un conteo de 1 a 6 utilizando la instrucción *do*. Luego compare el programa con las versiones de conteo con *for* y *while* elaboradas anteriormente.

```
#include "iostream"

using namespace std;

int main(void)
{
	int i;

	cout << "CONTAR DE 1 A 6" << endl << endl;

	i = 1;
	do
	{
		cout << i << endl;
		i = i+ 1;
	}
	while (i <= 6);

	cout << endl;
	return 0;
}
```

2) Elabore un programa que calcule el MCD de dos número utilizando el método de Euclides. Asuma inicialmente dos variables, una que contendrá el número mayor y la otra, el número menor. Luego proceda con este método así: se realiza la división entera del mayor entre el menor, si el residuo no es cero, el valor del menor se le asigna al mayor y el valor del residuo se le asigna al menor y se vuelve a dividir. Si el residuo es cero, el proceso se detiene y el resultado está en la variable del menor.

```
#include "iostream"

using namespace std;

int main(void)
{
	int mayor, menor, residuo;

	cout << "MCD DE DOS NÚMEROS\n\n";

	cout << "Digite el número mayor: ";
	cin >> mayor;
	cout << "Digite el número menor: ";
	cin >> menor;

	do
	{
		residuo = mayor % menor;
		if(residuo != 0){
			mayor = menor;
			menor = residuo;
		}
	}
	while(residuo != 0);
```

```
    cout << "El MCD es: " << menor;

    cout << endl;
    return 0;
}
```

En este problema puede notar que se debe obtener un residuo antes de revisarlo para verificar si se itera nuevamente, así que el lazo que se ajusta a este problema es un *do*. En el interior del lazo hay que hacer el intercambio iterativamente. Pero se hace condicionado para que, en la última iteración, no se le asigne cero a la variable llamada *menor*. Este problema también podría resolverse con lazo *while* y haciendo los ajustes correspondientes al programa, si es requerido. Lo que no se puede hacer es implementar un *for* porque no se puede determinar de antemano la cantidad de divisiones e intercambios que será necesario hacer.