

# Práctica 4. Estructuras de control iterativas

---

## Objetivos:

- Conocer y manejar con habilidad los distintos tipos de estructuras de control iterativas.

## 1. Sentencias de control iterativas

📖 No olvides que antes de pasar a la fase de diseño deberemos analizar exhaustivamente el problema.

**Las sentencias de control iterativas son aquellas que nos permiten variar o alterar la secuencia normal de ejecución de un programa haciendo posible que un grupo de acciones se ejecute más de una vez de forma consecutiva.** Este tipo de instrucciones también recibe el nombre de **bucle**. Cada vez que se ejecuta el cuerpo del bucle es una **iteración**.

A la hora de introducir bucles en nuestros programas debemos contestar a dos preguntas:

- **¿Qué se debe ejecutar en el cuerpo del bucle?** Debemos pensar qué sentencias son las que queremos que se repitan un número determinado de veces, estas son las sentencias que pondremos dentro del bucle.
- **¿Cuántas veces debe ejecutarse el cuerpo del bucle?** Debemos pensar en el número de veces que queremos que se ejecute el bucle. Este número de veces está controlado por una condición (una expresión lógica).

Según donde tenga la condición que controla el número de veces que se ejecuta el bucle, podemos distinguir 2 tipos de bucles:

- **Bucles con condición inicial.**
  - Sentencia **while**. Se repite mientras la condición sea verdadera. Este bucle se puede repetir **0 o más veces**.

```
while(expresión_lógica){  
    secuencia de sentencias;  
}
```

- Sentencia **for** (repetición con contador). Permite repetir su cuerpo un número determinado de veces. Este número de veces viene determinado por una variable contador.

```
for(inicializa_contador; expresión_lógica; incremento_contador){  
    secuencia de sentencias;  
}
```

- **Bucles con condición final.**

- Sentencia **do-while**. Se repite mientras la condición sea verdadera. Este bucle se repite **1 o más veces**.

```
do{  
    secuencia de sentencias;  
}while(expresión_lógica);
```

En los bucles se suelen utilizar algunas variables para unas tareas específicas, tales como contadores, acumuladores o interruptores.

### **Contadores:**

Un contador no es más que una variable destinada a contener un valor que se irá incrementando o decrementando en una cantidad fija. Se suelen utilizar para el control de procesos repetitivos.

### **Acumuladores:**

Un acumulador es una variable destinada a contener distintas cantidades provenientes de los resultados obtenidos en operaciones aritméticas previamente analizadas de manera sucesiva, lo que nos permitirá obtener el total acumulado de dichas cantidades. A diferencia de los contadores, no controlan los procesos repetitivos. Su inicialización depende de en qué operación matemática van a ser utilizados.

### **Interruptores (switches):**

Los interruptores, también denominados conmutadores o indicadores, son variables que pueden tomar dos únicos valores considerados como lógicos y opuestos entre sí a lo largo de todo el programa (0 ó 1, 1 ó -1, Verdadero o Falso, on/off, etc.).

Su objetivo es recordar en un determinado lugar del programa una ocurrencia o suceso acaecido o no con antelación, o hacer que dos acciones diferentes se ejecuten alternativamente en un proceso repetitivo. También deben ser inicializados. No se debe abusar de su utilización cuando no sea necesario.

## 2. Ejercicio resuelto 1. Repasa

- Implementa un algoritmo que lea un conjunto de números hasta que se introduzca el valor cero y visualice el número de números pares e impares introducidos.

### 2.1 Solución propuesta

```
1  #include<iostream>
2  using namespace std;
3  main(){
4      //Declaración de variables
5      int n, pares, impares;
6
7      cout<<"Introduce un número --0 para finalizar--: ";
8      cin>>n;
9
10     //Inicialización de variables
11     pares = 0;
12     impares = 0;
13
14     // Bucle se repetirá mientras no introduzcan el valor 0
15     while (n>0) {
16         if(n%2 == 0)
17             pares ++;
18         else
19             impares ++;
20         cout<<"Introduce un número --0 para finalizar--: ";
21         cin>>n;
22     }
23
24     cout<<"Cantidad de pares introducidos  "<<pares<<"\n";
25     cout<<"Cantidad de impares introducidos  "<<impares<<"\n";
26 }
```

## 3. Ejercicio resuelto 2. Repasa

- Implementa un algoritmo que muestre los múltiplos de 3 comprendidos entre 1 y n, siendo n un número que se introducirá por teclado. El programa se repetirá hasta que el número introducido sea 0.

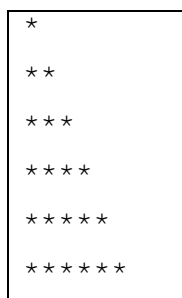
### 3.1 Solución propuesta

```
1  #include<iostream>
2  using namespace std;
3
4  main() {
5      int n, i;
6
7      cout<<"Introduce un número --0 para finalizar--: ";
8      cin>>n;
9      while(n>0){
10         cout<<"Los múltiplos de 3 entre 1 y "<<n<<" son: \n";
11         for(i=1; i<=n; i++) {
12             if(i%3 == 0)
13                 cout<<i<<endl;
14         }
15         cout<<"Introduce un número --0 para finalizar--: ";
16         cin>>n;
17     }
18 }
```

---

## 4. Ejercicio resuelto 3. Repasa

- Implementa un algoritmo que visualice por pantalla la siguiente figura, preguntando al usuario el número de estrellas que tiene que contener la línea final.



```
*
**
***
****
*****
*****
```

### 4.1 Solución propuesta

```
1  #include<iostream>
2  using namespace std;
3
4  main() {
5      int n, fil, col;
```

---

```
6
7     cout<<"Introduce el núm. de asteriscos de la línea final:
      ";
8     cin>>n;
9     for(fil = 1; fil <= n; fil++){
10         for(col = 1; col<=fil; col++){
11             cout<<"*";
12             cout<<endl; //Provoca un cambio de línea
13         }
14     }
```

## 5. Ejercicio resuelto 4. Repasa

- Escribe un programa que calcule  $X^N$  donde X es un número decimal y N es un entero positivo. El programa informará al usuario que el número N debe ser positivo si el usuario introduce un valor negativo. La solución propuesta no deberá utilizar la función pow().

Ejemplos:

```
pc-rosana:pruebasC rosana$ ./ej
Introduce X: 1.3
Introduce N: 5
1.3 elevado a 5 es: 3.71293
pc-rosana:pruebasC rosana$
```

```
pc-rosana:pruebasC rosana$ ./ej
Introduce X: 5.6
Introduce N: -3
N debe ser un número entero positivo
pc-rosana:pruebasC rosana$
```

### 5.1 Solución propuesta

```
1     #include<iostream>
2     using namespace std;
3
4     //Función principal
```

```
5    int main(){
6        float X, aux;
7        int N;
8
9        //Lee los dos valores necesarios
10       cout<<"Introduce X: ";
11       cin>>X;
12       cout<<"Introduce N: ";
13       cin>>N;
14       //Comprueba si N es correcto
15       if(N<=0){
16           cout<<"N debe ser un número entero positivo"<<endl;
17       }else{
18           //Multiplica N veces X
19           aux = X;
20           for(int i=N; i>1; i--){
21               aux = aux * X;
22           }
23           //Muestra el resultado
24           cout<<X<<" elevado a "<<N<<" es: "<<aux<<endl;
25       }
26   }
```

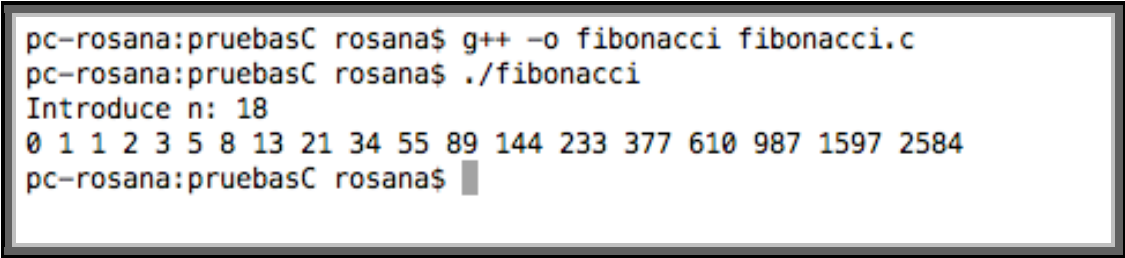
---

## 6. Ejercicio resuelto 5. Repasa

---

- Escribe un programa que pida un número  $n$  por teclado e imprima los valores de la serie Fibonacci hasta ese número  $n$ . Recuerda que  $\text{Fibonacci}(0)=0$ ,  $\text{Fibonacci}(1)=1$ , y  $\text{Fibonacci}(N)$  se calcula como  $\text{Fibonacci}(n-1)+\text{Fibonacci}(n-2)$ .

Ejemplo de ejecución:



```
pc-rosana:pruebasC rosana$ g++ -o fibonacci fibonacci.c
pc-rosana:pruebasC rosana$ ./fibonacci
Introduce n: 18
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
pc-rosana:pruebasC rosana$
```

### 6.1 Solución propuesta

```
1    #include<iostream>
```

---

```
2    using namespace std;
3
4    //Función principal
5    int main(){
6        int n;
7        int fibn1, fibn2, fibn;
8
9        cout<<"Introduce n: ";
10       cin>>n;
11
12       fibn2 = 0; //Fibonacci(0) = 0
13       fibn1 = 1; //Fibonacci(1) = 1
14
15       //Muestra el Fibonacci de 0 y el Fibonacci de 1 por
           pantalla
16       cout<<fibn2<<" "<<fibn1<<" ";
17
18       //Bucle para mostrar el resto de números Fibonacci.
19       for(int i=0; i<n-1; i++){
20
21           fibn = fibn1 + fibn2;
22
23           cout<<fibn<<" ";
24           fibn2 = fibn1;
25           fibn1 = fibn;
26       }
27       cout<<endl;
28   }
```

## 7. Ejercicio propuesto 1. Resuelve

- Implementa un programa que lea números que el usuario introduzca por teclado. En primer lugar el programa preguntará cuántos números se van a introducir. A continuación se solicitarán e introducirán uno a uno los números. El programa tiene que imprimir la suma de todos los números introducidos e indicar cuál es el mayor y el menor de todos.

**Ejemplo:**

```
pc-rosana:pruebasC rosana$ ./ej
¿Cuántos números vas a introducir? 12
Introduce un número: 3
Introduce un número: 67
Introduce un número: 9
Introduce un número: 3
Introduce un número: 65
Introduce un número: 23
Introduce un número: 67
Introduce un número: 98
Introduce un número: 1
Introduce un número: 0
Introduce un número: 23
Introduce un número: 54
La suma es: 413
El menor es: 0
El mayor es: 98
pc-rosana:pruebasC rosana$
```

## 8. Ejercicio propuesto 2. Resuelve

---

- Modifica el ejercicio anterior de forma que se desconocen el máximo de números a introducir y se detendrá cuando se introduzca un 0.

Ejemplo:

```
pc-rosana:pruebasC rosana$ ./ej
Introduce un número (0 para terminar) :56
Introduce un número (0 para terminar) :3
Introduce un número (0 para terminar) :89
Introduce un número (0 para terminar) :4
Introduce un número (0 para terminar) :14
Introduce un número (0 para terminar) :0
La suma es: 166
El menor es: 0
El mayor es: 89
pc-rosana:pruebasC rosana$
```

## 9. Ejercicio propuesto 3. Resuelve

---

- Implementa un programa en C que pida por teclado un número n, que deberá cumplir que sea mayor que 0 y menor o igual a 9. En caso de que la entrada de



dicho dato sea incorrecta, el programa deberá volver a solicitarlo hasta que sea correcto.

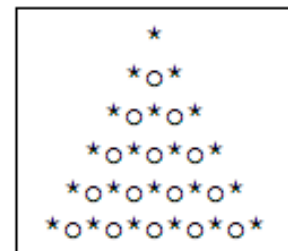
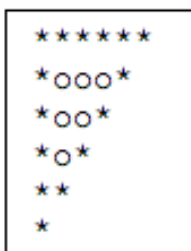
Después se deberá imprimir por pantalla todas las posibles parejas de valores (i,j), hasta (n,n).

Ejemplo:

```
pc-rosana:pruebasC rosana$ ./ej
Introduce N (0<N<=9): -1
El valor introducido es incorrecto. Vuelve a introducirlo
Introduce N (0<N<=9): 10
El valor introducido es incorrecto. Vuelve a introducirlo
Introduce N (0<N<=9): 6
(1,1) (1,2) (1,3) (1,4) (1,5) (1,6)
(2,1) (2,2) (2,3) (2,4) (2,5) (2,6)
(3,1) (3,2) (3,3) (3,4) (3,5) (3,6)
(4,1) (4,2) (4,3) (4,4) (4,5) (4,6)
(5,1) (5,2) (5,3) (5,4) (5,5) (5,6)
(6,1) (6,2) (6,3) (6,4) (6,5) (6,6)
pc-rosana:pruebasC rosana$
```

## 10. Ejercicio propuesto 4. Resuelve

- Implementa un programa que visualice en pantalla cada una de las siguientes figuras, preguntando al usuario el número de filas  $n$  que deben tener las figuras (En el ejemplo mostrado,  $n = 6$ ). Ten en cuenta que la solución que propongamos debe contener sentencias de salida por pantalla en las que se impriman exclusivamente uno solo de los siguientes caracteres: blanco ' ', asterisco '\*' y letra 'o'.



```
pc-rosana:pruebasC rosana$ ./ej
Introduce el número de filas: 6
*****
*000*
*00*
*0*
**
*

-----

0*****0
*****
*****
***
**
0

-----

      *
     *0*
    *0*0*
   *0*0*0*
  *0*0*0*0*
 *0*0*0*0*0*
pc-rosana:pruebasC rosana$
```

📖 **Recomendación:** Resuelve primero el ejercicio sin distinguir en las figuras los caracteres '\*' y 'o', es decir, dibuja sólo cada figura con el carácter '\*'. Después modifica tu solución inicial para incluir también en las figuras el carácter 'o'.

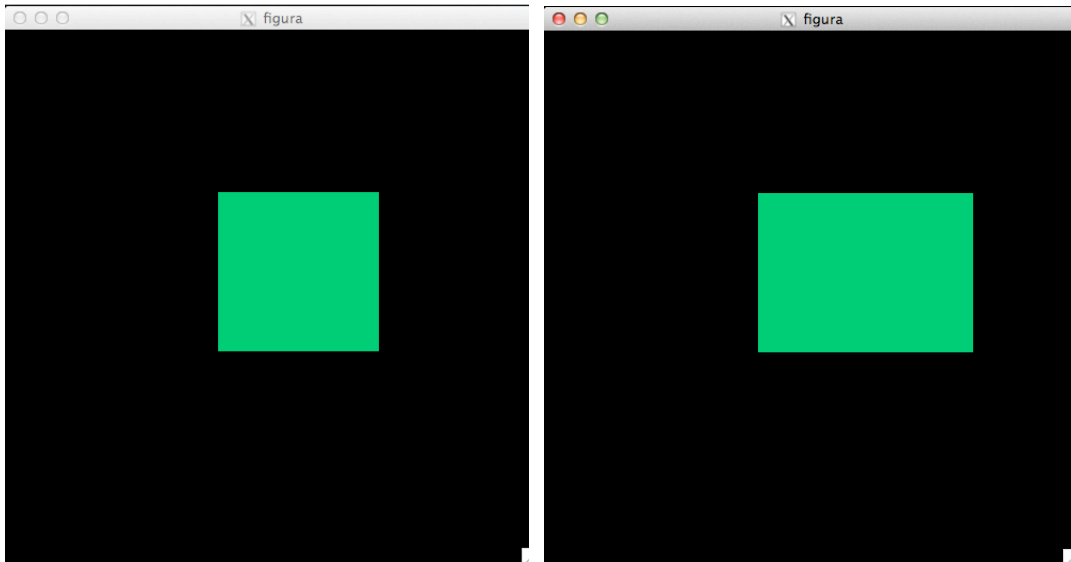
## 11. Ejercicio propuesto 5. Resuelve

---

- Recupera el ejercicio propuesto 5 del tema 3 (dibujaba un cuadrado o un rectángulo). Ayudándote de dicho ejercicio escribe un programa que dibuje el cuadrado o el rectángulo totalmente relleno.

```
MacBook-Pro-de-Rosana:pruebas rosanasatorre$ ./ej
¿Quieres dibujar un cuadrado (c) o un rectángulo (r) ? c
Introduce el tamaño del lado de tu cuadrado : 150
```

```
MacBook-Pro-de-Rosana:pruebas rosanasatorre$ ./ej
¿Quieres dibujar un cuadrado (c) o un rectángulo (r) ? r
Introduce el tamaño de uno de los lados de tu rectángulo : 150
Introduce el tamaño del otro de los lados de tu rectángulo : 200
```



## 11. Errores de programación comunes

**1. Bucles infinitos.** A veces no se controla bien la condición de control y se ejecuta un bucle infinito. Por ejemplo:

```
contador = 5;
while (contador != 0)
    contador = contador -2;
```

La variable contador tomará los valores 5, 3, 1, -1,... y estaríamos en un bucle infinito, puesto que nunca llega a tomar el valor 0.

**2. Uso de la misma variable de control en bucles anidados.** Habitualmente, cuando se programa se usan variables del tipo i, j, k... para controlar bucles anidados. En bloques anidados es frecuente que se olvide la variable de control y se reutilice una ya usada en bucles más externos o usada para otro propósito. Para evitar esto, conviene reservar ese tipo de variables exclusivamente para el control de bucles y no usar variables poco descriptivas (como *contador*) para controlar los bucles.

**3. Suponer que el compilador asigna valor cero a las variables sin inicializar.** Si se escribe:

```
int i;  
  
while (i<10)...
```

Se está asumiendo que el valor inicial de i es 0. Esto no tiene por qué ser cierto.  
**Nunca se deben hacer ese tipo de suposiciones y se deben inicializar todas las variables.**

**4. Errores de tipo fallo por uno en número de iteraciones.** Si se quiere ejecutar un bucle controlado por contador n veces. Probamos con este código, **¿qué sucede?**

```
contador = 1;  
  
while (contador<n){  
    contador++;  
}
```

**Este bucle sólo se ejecuta n-1 veces.**

Vamos a probar entonces, con este código, **¿qué sucede?**

```
contador = 0;  
  
while(contador<=n){  
    contador++;  
}
```

**El bucle se ejecuta n+1 veces.**

**Son errores muy fáciles de cometer y muy difíciles de detectar. ¿Cómo se debería hacer?**

```
contador = 0;  
  
while(contador<n){
```

```
        contador++;  
    }
```

**o bien, también sería correcto:**

```
    contador = 1;  
    while(contador<=n){  
        contador++;  
    }
```

**5. Confundir el criterio de comparación del bucle *for*.** Si se escribe:

```
for (i=0; i<10; i++)
```

Se ejecuta el contenido del bucle 10 veces. Si en lugar de eso escribimos:

```
for(i=0; i==10; i++)
```

no ocurre lo mismo. ¿**Qué ocurre ahora?**

Se ejecuta la primera iteración y se compara la condición de control ( $i = 10$ ). Como la condición no es cierta, el bucle se terminaría.

## ANEXO. Ejemplos de código en Java

### Código del ejercicio propuesto 1

**Implementa un algoritmo que lea un conjunto de números hasta que se introduzca el valor cero y visualice el número de números pares e impares introducidos.**

```
import java.util.Scanner;
```

```
public class ejercicio1{
```

```
    public static void main(String[] args)
```

```
{  
    int n, pares, impares;  
  
    Scanner sc = new Scanner(System.in);  
  
    System.out.print("Introduce un número --0 para finalizar--: ");  
    n = sc.nextInt();  
  
    pares = 0;  
    impares = 0;  
  
    while(n>0){  
        if(n%2==0)  
            pares++;  
        else  
            impares++;  
  
        System.out.print("Introduce un número --0 para finalizar--: ");  
        n = sc.nextInt();  
    }  
  
    System.out.println("La cantidad de pares introducidos es "+pares);  
    System.out.println("La cantidad de impares introducidos es "+impares);  
}
```

### **Código del ejercicio propuesto 2**

**Implementa un algoritmo que muestre los múltiplos de 3 comprendidos entre 1 y n, siendo n un número que se introducirá por teclado. El programa se repetirá hasta que el número introducido sea 0.**

```
import java.util.Scanner;
```

```
public class ejercicio2{

    public static void main(String[] args)
    {
        int n, i;

        Scanner sc = new Scanner(System.in);

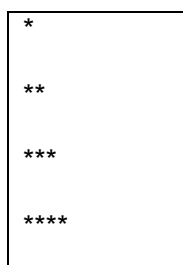
        System.out.print("Introduce un número --0 para finalizar--: ");
        n = sc.nextInt();

        while(n>0){
            System.out.println("Los múltiplos de 3 entre 1 y "+n+" son:");
            for(i=1;i<=n;i++){
                if(i%3 == 0)
                    System.out.println(i);
            }

            System.out.print("Introduce un número --0 para finalizar--: ");
            n = sc.nextInt();
        }
    }
}
```

### **Código del ejercicio propuesto 3**

**Implementa un algoritmo que visualice por pantalla la siguiente figura, preguntando al usuario el número de estrellas que tiene que contener la línea final.**



```
*
**
***
****
```

```
*****
```

```
*****
```

```
import java.util.Scanner;
```

```
public class ejercicio3{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        int n, fil,col;
```

```
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Introduce el número de asteriscos de la línea final: ");
```

```
        n = sc.nextInt();
```

```
        for(fil=1; fil<=n; fil++){
```

```
            for(col=1; col<=fil; col++){
```

```
                System.out.print("*");
```

```
            System.out.println(" ");
```

```
        }
```

```
    }
```

```
}
```

### **Código del ejercicio propuesto 4**

```
import java.util.Scanner;
```

```
public class ejemplo4{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        int N;
```

```
        float X, aux;
```



```
Scanner sc = new Scanner(System.in);

//Lee los valores necesarios
System.out.print("Introduce X: ");
X = sc.nextFloat();

System.out.print("Introduce N: ");
N = sc.nextInt();

//Comprueba si N es correcto
if(N<=0){
    System.out.println("N debe ser un número entero positivo");
}else{
    //Multiplica N veces X
    aux = X;
    for(int i=N; i>1; i--){
        aux = aux * X;
    }

    //Muestra el resultado
    System.out.println(X+" elevado a "+N+" es: "+aux);
}
}
}
```