Estructuras cíclicas I

Mientras y Hacer Mientras - *While* y *Do While*

Elizabeth León Guzmán, Ph.D.
eleonguz@unal.edu.co

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Carlos Andres Sierra, M.Sc.
casierrav@unal.edu.co

Research Group on Data Mining – Grupo de Investigación en Minería de Datos – (Midas)
Research Group on Artificial Life – Grupo de Investigación en Vida Artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Recordatorio/Aclaración

Por claridad, todas las funciones que desarrollaremos en esta sesión se programarán como parte de la clase `Main`.

```
import java.util.Scanner;
public class Main {
  /**
   * Aqui van las funciones a desarrollar en esta sesión
   */
  // Función principal
  public static void main(String[] args){
    /**
     * Aquí va el código para probar las funciones creadas
     */
  }
}
```

Agenda

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

El ciclo mientras (`while`) I

El ciclo **mientras** (`while`) permite ejecutar un bloque de instrucciones mientras que una expresión booleana dada se cumpla, es decir, mientras su evaluación dé como resultado verdadero.

La expresión booleana se denomina **condición de parada** y siempre se evalúa antes de ejecutar el bloque de instrucciones; tras esto se pueden presentar dos casos:

- Si la condición no se cumple, el bloque no se ejecuta.
- Si la condición se cumple, el bloque se ejecuta, después de lo cual la instrucción vuelve a empezar, es decir, la condición se vuelve a evaluar.

El ciclo mientras (while) II

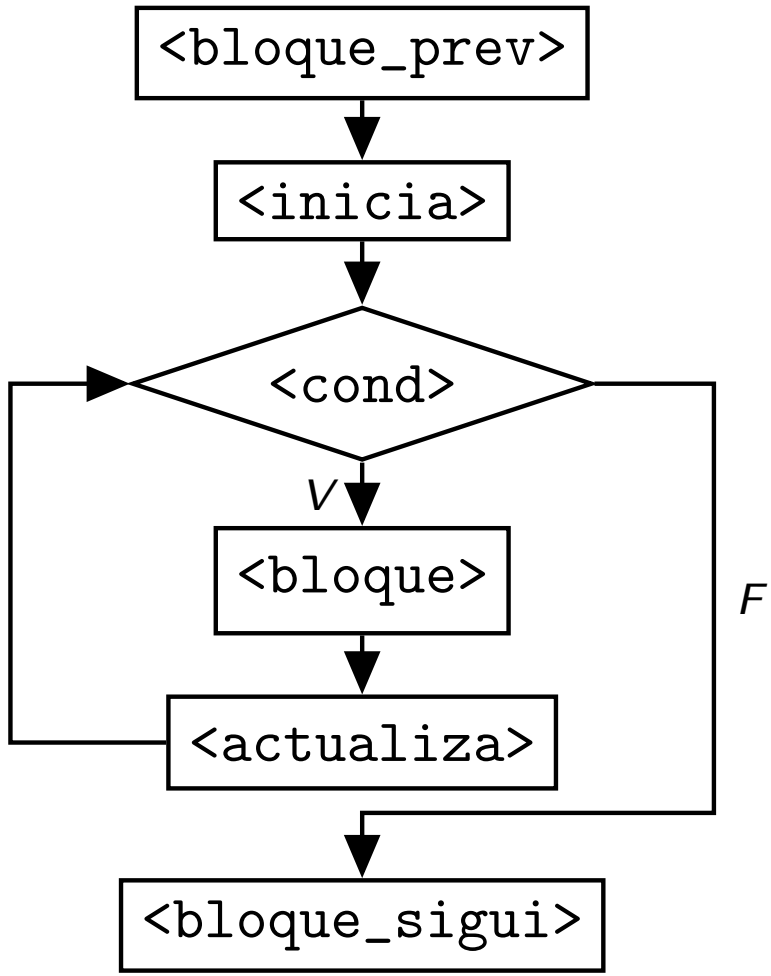
En el caso en que la condición se evalúe la primera vez como falsa, el bloque de instrucciones no será ejecutado, lo cual quiere decir que el número de repeticiones o iteraciones de este bloque será cero (0). Si la condición siempre evalúa a verdadero, la instrucción se ejecutará indefinidamente, es decir, un número infinito de veces.

De esta manera, una iteración es una ejecución del bloque de instrucciones interno del ciclo.

MIDAS

El ciclo mientras (while) III

Un ciclo **mientras** (while) se puede representar gráficamente mediante un diagrama de flujo de la siguiente manera.



El ciclo mientras (while) IV

Un esquema textual que en Java representa un ciclo mientras (while) es la que se da en el siguiente fragmento de código.

```
<bloque_prev>
<inicia>
while (<cond>) {
   <bloque>
   <actualiza>
}
<bloque_sigui>
```

No olvidar que el ámbito (alcance) del ciclo mientras es lo que esté encerrado dentro de los corchetes que se encuentran después de la verificación de la condición de parada.

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

El ciclo mientras (while) V

Las partes del ciclo mientras (while) son:

- El fragmento <bloque_prev> es el bloque de instrucciones previas que han sido ejecutadas antes del ciclo.
- El fragmento <inicia> es el bloque de instrucciones donde se inicializan las variables que intervienen en la condición de parada.
- El fragmento <cond> es la condición de parada que se evalúa cada vez que se inicia o se reinicia el ciclo.

MIDAS

El ciclo mientras (while) VI

- El fragmento <bloque> es el bloque de instrucciones principal del ciclo que se ejecuta mientras la condición se cumpla.
- El fragmento <actualiza> es el bloque que se utiliza para actualizar las variables que son utilizadas para evaluar la condición de parada cuando se intenta reiniciar el ciclo.
- El fragmento <bloque_sigui> es el bloque de instrucciones que se ejecutan después de terminar de ejecutar el ciclo.

Ejemplo 1 del ciclo mientras (while) I

Ejemplo

Para el siguiente fragmento de código que contiene un ciclo mientras (while)

```
<bloque_prev>
int i = 0;
while (i <= 6) {
    System.out.println(i);
    i = i + 1;
}
<bloque_sigui>
```

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Ejemplo 1 del ciclo mientras (while) II

Ejemplo (continuación)

se tiene que en el fragmento del código:

<inicia> corresponde a la instrucción

```
int i = 0;
```

<cond> corresponde a la instrucción

```
i <= 6
```

<bloque> corresponde a la instrucción

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

<actualiza> corresponde a la instrucción

```
i = i + 1;
```

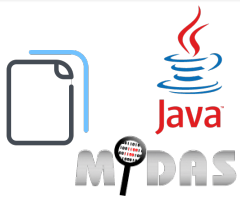
Ejemplo 1 del ciclo mientras (while) III

Ejemplo (continuación)

cuando se ejecuta este ciclo, lo que se obtiene en la consola de salida es el texto que se presenta en el cuadro a la derecha

```
int i = 0;
while (i <= 6) {
    System.out.println(i);
    i = i + 1;
}
```

```
...
0
1
2
3
4
5
6
...
```



Ejemplo 1 del ciclo mientras (while) IV

Ejemplo (continuación)

en este caso la salida que se produce es la anterior porque el bloque

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

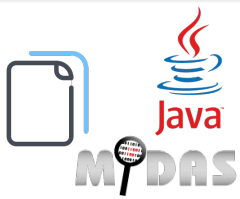
se ejecuta siete veces, variando `i` desde 0 hasta cuando `i` toma el valor 7, que hace que la condición se evalué falso y por lo tanto termina el ciclo; obsérvese que la variable termina el ciclo con valor $i = 7$, pero este valor no se imprime pues para este caso la condición se evalúo falso.

Ejemplo 2 del ciclo mientras (while) I

Ejemplo

Para el siguiente fragmento de código que contiene un ciclo while

```
int i = 2;                                  // inicializa a i en 2
int j = 25;                                 // inicializa a j en 25
while (i < j) {                        // mientras i sea menor a j
   // imprime los valores de i y j
   System.out.println(i + ", " + j);
   i = i * 2;              // i se multíplica por 2 en cada paso
   j = j + 10;                          // se incrementa de 10 en 10
}
// se ejecuta al terminar el ciclo
System.out.println("the end.");
System.out.println(i + ", " + j);// valores finales de i y j
```



Ejemplo 2 del ciclo mientras (while) II

Ejemplo (continuación)

las variables `i` y `j` se inicializan con los valores 2 y 25 respectivamente, luego se verifica que 2 sea menor estrictamente que 25, a continuación se imprime el valor de la variable `i` seguido por una coma y un espacio (", "), seguido por el valor de la variable `j`, seguido de un salto de línea; a continuación se multiplica la variable `i` por 2 y a la variable `j` se le suma 10. Esto se realiza hasta que el valor de la variable `i` sea mayor o igual a el valor de la variable `j`.

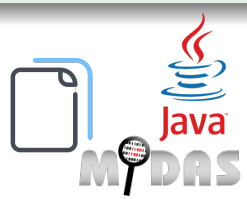
Ejemplo 2 del ciclo mientras (while) III

Ejemplo (continuación)

El resultado de la ejecución de este ciclo mostrado en la consola de salida es el texto que se presenta en el cuadro a la derecha.

```
int i = 2;
int j = 25;
while (i < j) {
    System.out.print(i + ", " + j);
    i = i * 2;
    j = j + 10;
}
System.out.println("the end.");
System.out.print(i + ", " + j);
```

```
...
2, 25
4, 35
8, 45
16, 55
32, 65
64, 75
the end.
128, 85
...
```



Ejemplo 2 del ciclo mientras (while) IV

Ejemplo (continuación)

Obsérvese que las variables `i` y `j` terminan el ciclo con los valores 128 y 85, y como 128 no es menor que 85, entonces el ciclo se para y se sale de éste, por esta razón no se imprimen estos valores dentro del ciclo, sólo se imprimen luego de salir del ciclo.

Ejemplo 3. El mínimo número positivo de la máquina I

Ejemplo (el mínimo número positivo de la máquina)

Dado que los números reales que son representables en un computador son finitos, entonces es posible hablar del menor número positivo representable en la máquina, es decir el número

$$x_{\text{min}} = \min\left\{x : (x \text{ es un número de máquina}) \wedge (x > 0)\right\}$$

Para encontrar dicho número hay un algoritmo muy sencillo que permite encontrar el valor.

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Ejemplo 3. El mínimo número positivo de la máquina II

Ejemplo (continuación)

Teniendo en cuenta que los números en un computador se representan usando base 2, un algoritmo que permite hallar el mínimo número positivo de la máquina consiste en calcular los términos de una progresión geométrica que inicia con el término $x_0 = 1$ y para la cual los términos siguientes se calculan utilizando la razón de la progresión $r = 1/2$, es decir, $x_{n+1} = \frac{x_n}{2}$, esto se realiza mientras cada nuevo término es positivo.

La codificación en Java de una función constante que permite hallar el mínimo número positivo representable en la máquina junto con su programa principal se presenta a continuación

Ejemplo 3. El mínimo número positivo de la máquina III

Ejemplo (continuación)

```
public class Main {
    public static double minMaquina() {
        double Xo = 1.0;
        double Xi = Xo / 2.0;
        while (Xi > 0.0) {
            Xo = Xi;
            Xi = Xo / 2.0;
        }
        return Xo;
    }
    public static void main(String[] args) {
        System.out.println(minMaquina());
    }
}
```

Ejemplo 3. El mínimo número positivo de la máquina IV

Ejemplo (continuación)

Si se ejecuta el anterior programa, el resultado que se obtiene es el siguiente

```
4.9E-324
```

como para calcular cada término de la progresión geométrica se necesita únicamente el término anterior, entonces son necesarias sólo dos variables, las cuales se utilizarán de la siguiente manera:

- La variable `Xo` representa el término x_n y se inicializa con el valor $x_0 = 1.0$.
- La variable `Xi` representa el término x_{n+1} y se inicializa con el valor `Xo / 2.0`.

Ejemplo 3. El mínimo número positivo de la máquina V

Ejemplo (continuación)

- Ahora, dentro del ciclo la variable `Xo` jugará el rol del término x_{n+1} mediante la asignación `Xo = Xi`.
- A la variable `Xi` se le asigna el siguiente término calculado de la progresión y es asignado mediante la expresión `Xi = Xo / 2.0`.
- Las dos últimas rutinas descritas se realizan mientras el valor de la variable `Xi` sea mayor a `0.0`.
- El ciclo en algún momento deja de ejecutarse[a] ya que el conjunto de los números de máquina es finito, la progresión geométrica es decreciente y está acotada inferiormente por `0`.
- Finalmente, se retorna el valor almacenado en la variable `Xo` pues fue el último término de la progresión que era distinto de `0`.

[a] Aunque matemáticamente se pueden calcular infinitos términos de la progresión.

Problemas

Problemas

1. Diseñe un algoritmo que involucre un ciclo y que nunca ingrese al ciclo.
2. Diseñe un algoritmo que involucre un ciclo y que se ejecute indefinidamente.
3. Diseñe un algoritmo que pida un valor entero, y que siga leyendo valores enteros mientras que alguno de esos valores no represente el código ASCII de una letra mayúscula del alfabeto inglés.

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Agenda

MIDAS

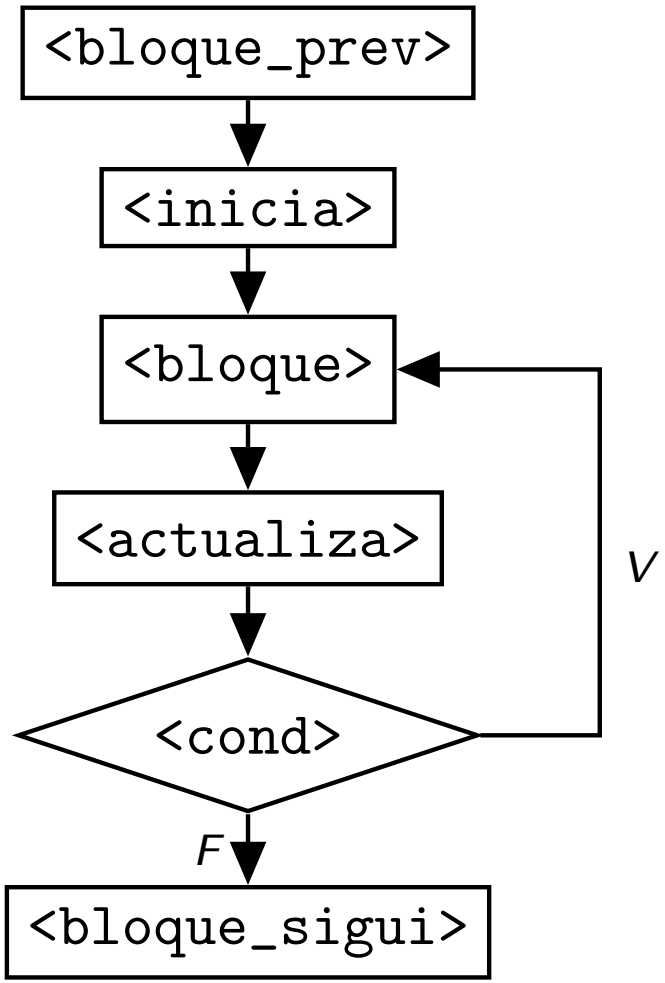
UNIVERSIDAD NACIONAL DE COLOMBIA un

El ciclo hacer-mientras (do) I

Existe otra estructura cíclica en programación, ésta se conoce como un ciclo **hacer-mientras** (`do`). Esta estructura es casi equivalente a la estructura mientras (`while`), ya que usualmente se utiliza cuando con seguridad y de forma anticipada se sabe que se hará al menos una evaluación del bloque principal del ciclo. En esta estructura cíclica la verificación de la condición de parada se realiza al final del ciclo.

El ciclo hacer-mientras (do) II

Un ciclo **hacer-mientras** (do) se puede representar gráficamente mediante un diagrama de flujo de la siguiente manera.



El ciclo hacer-mientras (do) III

Un esquema textual que en Java representa un ciclo do es la que se da en el siguiente fragmento de código, obsérvese que en un ciclo do al menos una vez se ejecuta el bloque principal del ciclo y la actualización.

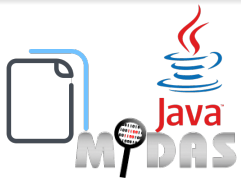
```
<bloque_prev>
<inicia>
do {
    <bloque>
    <actualiza>
} while (<cond>);
<bloque_sigui>
```

El mínimo número positivo de la máquina versión do

Ejemplo

Un algoritmo para obtener el mínimo número positivo de la máquina que usa un ciclo hacer-mientras (do) se muestra a continuación

```
public static double minMaquina() {
    double Xo = 1.0;
    double Xi = Xo / 2;
    do {
        Xo = Xi;
        Xi = Xo / 2.0;
    } while (Xi > 0.0);
    return Xo;
}
```



Agenda

Terminación forzada de un ciclo I

Cuando se construyen algoritmos que utilizan ciclos, algunos programadores requieren que dadas unas condiciones dentro del bloque de instrucciones internas del ciclo, se pare la ejecución del ciclo y se continue con las instrucciones subsiguientes.

Este tipo de salidas forzadas, se obtiene usando la instrucción `break`. El uso de ésta no se recomienda, pues hace difícil realizarle un seguimiento al programa, pues dejan muchas variables en valores que no se pueden controlar y/o determinar apropiadamente.

MIDAS

Terminación forzada de un ciclo II

Sin embargo, la instrucción `break` es utilizada por programadores ya sea por facilidad (algunos no realizan un buen diseño de los algoritmos cíclicos), a veces por velocidad de ejecución (sin verificar las condiciones de continuación o terminación del ciclo), ésta última situación es muy común cuando se usan ciclos de tipo para (`for`).

Ejemplo

Desarrollar un programa que lea números enteros y los sume hasta que lea un cero (0). Un algoritmo que soluciona este problema es el que se muestra a continuación

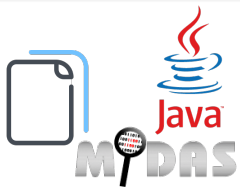
MIDAS

Terminación forzada de un ciclo III

Ejemplo (continuación)

```
public static void main(String[] args){
  int suma = 0;
  while (true) {
    System.out.print("Ingrese un número entero ");
    System.out.println("a sumar o 0 para salir: ");
    int dato = Integer.parseInt(sc.nextLine());
    if (dato == 0) break;
    suma += dato;
  }
  System.out.println("La suma es: " + suma);
}
```

El break se utiliza para salir cuando el usuario ingresa un cero (0).



Eliminación de la instrucción `break` I

Todo programa con un ciclo mientras (`while`) que tenga una instrucción `break` se puede ajustar para que no la use, sin embargo, el proceso de eliminar dicha instrucción requiere leer y entender correctamente el papel de la instrucción `break` en el ciclo. Cuando se usa por velocidad en los ciclos para (`for`), es muy difícil (sino imposible) eliminarla.

MIDAS

Eliminación de la instrucción `break` II

Ejemplo (Eliminar la instrucción `break` del programa del ejemplo anterior)

Lo primero que se puede determinar es que se puede cambiar la condición del ciclo para que deje de ser siempre verdadera por una que dependa del valor del dato leído (que es lo que se usa para ejecutar la instrucción `break`). Lo otro importante es que se debe leer al menos una vez un dato, ya sea para no leer más datos (si el usuario ingresa 0), o para sumarlos. De esta manera se puede usar el ciclo hacer-mientras (`do`). Por lo tanto, la eliminación del `break` produce el algoritmo que se muestra a continuación

MIDAS

Eliminación de la instrucción break III

Ejemplo (continuación)

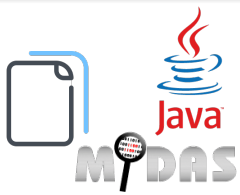
```
public static void main(String[] args){
  int dato = 0;
  int suma = 0;
  do {
    System.out.print("Ingrese un número entero ");
    System.out.println("a sumar o 0 para salir: ");
    dato = Integer.parseInt(sc.nextLine());
    suma += dato;
  } while (dato != 0);
  System.out.println("La suma es: " + suma);
}
```

Eliminación de la instrucción break IV

Ejemplo (continuación)

Se puede simplificar el programa, si se entiende muy bien el algoritmo.

```
public static void main(String[] args){
  int dato = 1;
  int suma = 0;
  while (dato != 0) {
    System.out.print("Ingrese un número entero ");
    System.out.println("a sumar o 0 para salir: ");
    dato = Integer.parseInt(sc.nextLine());
    suma += dato;
  }
  System.out.println("La suma es: " + suma);
}
```



Agenda

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Teorema fundamental de la programación estructurada

En 1966 C. Böhm y G. Jacopini demostraron el siguiente teorema

Teorema (Teorema fundamental de la programación estructurada)

Un lenguaje de programación es completo en Turing siempre que tenga variables enteras no negativas, las operaciones aritméticas elementales sobre dichas variables, y que permita ejecutar enunciados en forma secuencial, incluyendo enunciados de asignación (=), selección (if) y ciclos (while).

Lo que significa que en este momento

¡ya es posible programar cualquier algoritmo que pueda ser ejecutado en un computador¡.

Problemas varios I

Problemas

1. Imprimir un listado con los números del 1 al 100 cada uno con su respectivo cuadrado.
2. Imprimir un listado con los números impares desde 1 hasta 999 y seguidamente otro listado con los números pares desde 2 hasta 1000.
3. Imprimir los números pares en forma descendente hasta 2 que son menores o iguales a un número natural $n \geq 2$ dado.
4. En 2022 el país A tendrá una población de 25 millones de habitantes y el país B de 18.9 millones. Las tasas de crecimiento anual de la población serán de 2% y 3% respectivamente. Desarrollar un algoritmo para informar en qué año la población del país B superará a la de A.

Problemas varios II

Problemas

5. Diseñar una función que permita calcular el épsilon de la máquina. El épsilon de máquina es el número decimal más pequeño que sumado a 1 se puede representar de manera precisa en la máquina (que no es redondeado), es decir, retorna un valor diferente de 1, éste da una idea de la precisión o número de cifras reales que pueden ser almacenadas en la máquina. La idea es realizar un ciclo en el cual se realiza la operación $1 + \epsilon$ para potencias de 2 desde $\epsilon = 2^0$ y continuando con potencias decrecientes de 2 $\left(\epsilon = 2^{-1}, \epsilon = 2^{-2}, \epsilon = 2^{-3}, \epsilon = 2^{-4}, \ldots\right)$ hasta obtener que el resultado de la suma $1 + \epsilon$ no se altere.