Estructuras cíclicas II

Para - *For*

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

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Complementos a los ciclos para (for)
4. Los ciclos para (for) como iteradores de colecciones
5. Problemas

MIDAS

La estructura de control de ciclos para (for)

Existe otra estructura cíclica que es de uso frecuente en programación, el ciclo para (for). Esta estructura de control tiene dos propósitos primordiales que no siempre son soportados por todo lenguaje de programación:

1. Como una forma compacta de escribir un ciclo mientras (while).
2. Para *iterar* sobre los elementos de una colección de elementos.

Esta estructura es usualmente utilizada cuando se conocen los valores inicial y final de la variable que es utilizada en la condición de parada.

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Agenda

1 La estructura de control de ciclos para (for)

2 Los ciclos para como versiones compactas de ciclos mientras

3 Complementos a los ciclos para (for)

4 Los ciclos para (for) como iteradores de colecciones

5 Problemas

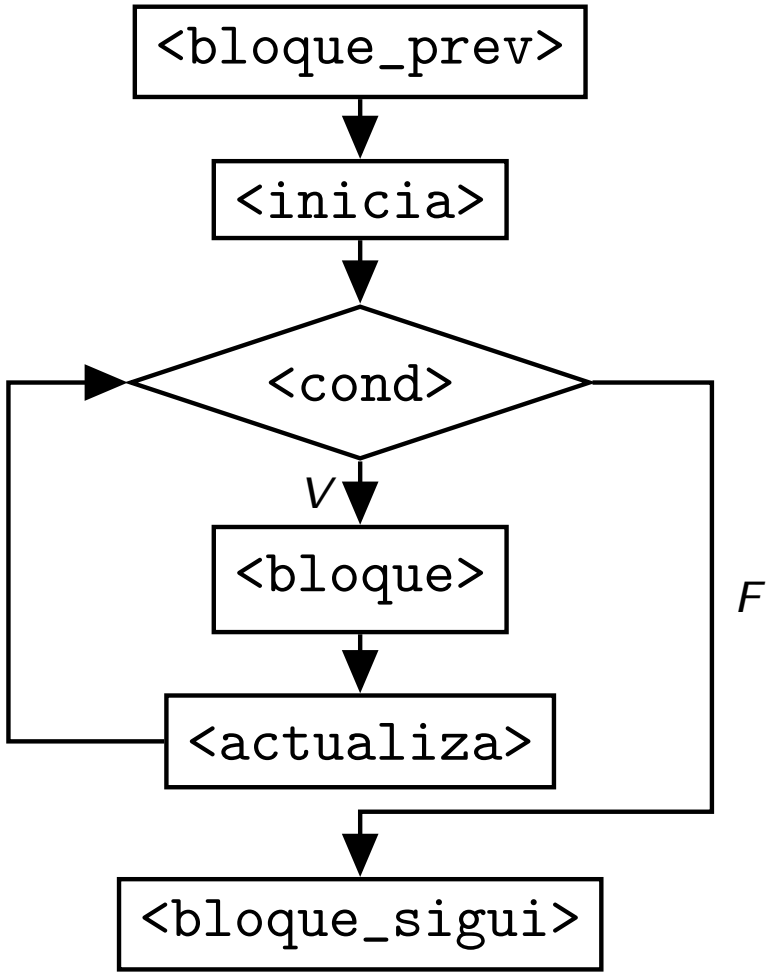
MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

El ciclo para como versión del ciclo mientras I

En muchos lenguajes (incluido Java), el ciclo para (`for`) se usa para escribir de manera compacta un ciclo `mientras` (`while`). La representación gráfica mediante un diagrama de flujo es la misma que la de un ciclo `mientras` (`while`), la cual es la siguiente:

El ciclo para como versión del ciclo mientras II



El ciclo para como versión del ciclo mientras III

La sintaxis general de un ciclo mientras (`while`) en el lenguaje de programación Java es:

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

El ciclo para como versión del ciclo mientras IV

En Java el esquema textual que representa un ciclo para (for) es el que se da en el siguiente fragmento de código. Se observa que un ciclo (for) es equivalente a un ciclo (while) pero su representación sintáctica es más compacta y separa el control del ciclo del cálculo que se desea realizar con el ciclo.

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

Ejemplo 1. Suma de los primeros n números naturales I

Ejemplo (La suma de los primeros n números naturales)

Las dos siguientes funciones permiten calcular la suma de los primeros n números naturales positivos, es decir, permiten calcular el valor de la expresión

$1 + 2 + 3 + \cdots + (n-1) + n$, que abreviadamante se escribe como $\sum_{i=1}^{n} i$

```
int suma(int n) {
    int s = 0;
    int i = 1;
    while (i <= n) {
        s = s + i;
        i++;
    }
    return s;
}
```

```
public static int suma(int n) {
    int s = 0;
    for (int i = 1; i <= n; i++) {
        s = s + i;
    }
    return s;
}
```

Ejemplo 1. Suma de los primeros *n* números naturales II

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

estas dos funciones son equivalentes, ya que ejecutan las mismas modificaciones de las variables, pues se tiene que el fragmento de código:

<bloque_prev> corresponde a la instrucción

```
int s = 0;
```

<inicia> corresponde a la instrucción

```
int i = 1;
```

<cond> corresponde a la instrucción

```
i <= n
```

Ejemplo 1. Suma de los primeros *n* números naturales III

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

`<bloque>` corresponde a la instrucción

```
s = s + i;
```

`<actualiza>` corresponde a la instrucción

```
i++;
```

`<bloque_sigui>` corresponde a la instrucción

```
return s;
```

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Ejemplo 1. Suma de los primeros *n* números naturales IV

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

En la construcción de estas funciones aparecen dos variables que tienen una connotación muy importante:

- La variable `i` juega el rol de una **variable contadora** ya que ésta permite llevar el conteo de cuántos ciclos se han efectuado.
- La variable `s` juega el rol de una **variable acumuladora** pues en ésta se acumula o almacena el valor parcial que se desea calcular utilizando el ciclo.

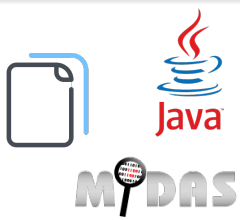
MIDAS

Ejemplo 1. Suma de los primeros *n* números naturales V

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

La codificación en Java de una función que permite sumar los primeros *n* números naturales positivos junto con su programa principal es

```
import java.util.Scanner;
public class Main {
   public static int suma(int n) {
      int s = 0;
      for (int i = 1; i <= n; i++) {
         s = s + i;
      }
      return s;
   }
```



Ejemplo 1. Suma de los primeros *n* números naturales VI

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Ingrese el valor de n: ");
        int n = sc.nextInt();
        System.out.print("La suma de los primeros " + n +
                                        " números naturales es: ");
        System.out.print(suma(n));
    }
}
```

Ejemplo 1. Suma de los primeros *n* números naturales VII

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

Si se ejecuta el anterior programa y como entrada se ingresa el valor `n` $= 6$ (por ejemplo el número de caras de un dado ⚀ + ⚁ + ⚂ + ⚃ + ⚄ + ⚅), el resultado que se obtiene es el siguiente

```
Ingrese el valor de n: 6
La suma de los primeros 6 números naturales es: 21
```

Ejemplo 1. Suma de los primeros *n* números naturales VIII

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

En general es fácil comprobar si el resultado que se obtiene utilizando la función `suma(n)` es correcto, pues existe una fórmula muy sencilla para calcular la suma de los primeros n números naturales positivos sin necesidad de realizar la suma exhaustiva. Esta fórmula es:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Agenda

1 La estructura de control de ciclos para (for)

2 Los ciclos para como versiones compactas de ciclos mientras

3 Complementos a los ciclos para (for)

4 Los ciclos para (for) como iteradores de colecciones

5 Problemas

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Complementos a los ciclos para (for) I

Puede existir más de una expresión de inicio que sea del mismo tipo así como también más de una expresión de actualización, esto se puede observar en el siguiente ejemplo

Ejemplo

```
for(int i = 0, j = 10; i <= j; i++, j--){
   System.out.println(i + ", " + j);
}
```

```
0, 10
1, 9
2, 8
3, 7
4, 6
5, 5
```

Complementos a los ciclos para (for) II

Es necesario tener muy en claro el comportamiento del ciclo, ya que una condición o una expresión de actualización mal diseñada podrían dar origen a un ciclo infinito. Ésto se observa en el siguiente ejemplo

Ejemplo

```
for (int i = 1; i <= 10; i--) {
    System.out.println(i);
}
```

```
1
0
-1
-2
-3
...
```

La variable `i` nunca llega a ser 10, de modo que el ciclo sigue ejecutándose indefinidamente.

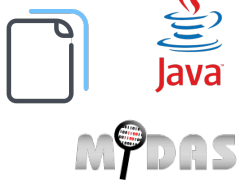
Finalizando un ciclo (for)

Tal y como se mencionó anteriormente, la expresión `break` resulta en la terminación forzada del ciclo, sin importar cuál sea la condición de terminación del ciclo. Ésto se observa en el siguiente ejemplo

Ejemplo (Finalizando un ciclo for)

```
for (int i = 0; i <= 30; i++) {
    if (i == 4) {
        break;
    }
    System.out.println(i);
}
```

```
0
1
2
3
```



Agenda

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Complementos a los ciclos para (for)
4. Los ciclos para (for) como iteradores de colecciones
5. Problemas

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Ciclos para (for) como iteradores de colecciones I

Un ciclo para (`for`) puede ser usado para obtener uno a uno los elementos de una colección de elementos y poder realizar con cada uno de ellos el mismo bloque de operaciones.

Un esquema textual que en Java representa dicho ciclo para (`for`) es el que se da en el siguiente fragmento de código.

```
<bloque_prev>
for ( T <elemento>:<coleccion> ) {
   <bloque>
}
<bloque_sigui>
```

Aquí `T` es el tipo de elementos de la colección.

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Ciclos para (for) como iteradores de colecciones II

en donde:

- El fragmento <bloque_prev> es el bloque de instrucciones previas que han sido ejecutadas antes del ciclo.
- El fragmento T <elemento> define la variable que se usa para ir recorriendo (iterando) sobre los elementos de la colección (tomará como valor cada uno de ellos en cada iteración).
- El fragmento <coleccion> es la colección de elementos que será recorrida (iterada) con el ciclo.

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Ciclos para (for) como iteradores de colecciones III

- El fragmento <bloque> es el bloque de instrucciones principal del ciclo que se ejecuta con cada uno de los elementos de la colección.
- El fragmento <bloque_sigui> es el bloque de instrucciones que se ejecutan después de terminar de ejecutar el ciclo.

El ciclo para (for) se puede leer en castellano como: "*Para cada* <elemento> *en la* <coleccion> *realice las instrucciones en el* <bloque>".

Recorriendo un arreglo I

Ejemplo

Dado el arreglo[†] `["Tomate de árbol", "Maracuyá ", "Guayaba"]`, un programa que permite imprimir todos sus elementos es:

```
public static void main(String[] args){
   String[] frutas =
      new String[]{"Tomate de árbol", "Maracuyá ", "Guayaba"};
   for(String f:frutas) { // para cada elemento f en la lista
      System.out.println(f);
   }
}
```

[†]Un arreglo es una colección de elementos que se puede definir en Java.

Recorriendo un arreglo II

Ejemplo (continuación)

El resultado de ejecutar este programa será entonces:

```
...
Tomate de árbol
Maracuyá
Guayaba
...
```

Forzando la detención del recorrido del arreglo I

Si se requiere detener el ciclo antes de iterar sobre toda la colección de elementos se debe usar la sentencia `break`.

Ejemplo (Recorriendo una lista hasta encontrar un elemento especial)

Dada la lista de frutas

```
["Pera", "Maracuyá ", "Guayaba", "Lulo", "Granadilla"]
```

imprimir sus elementos hasta encontrar la fruta `"Lulo"`.

MIDAS

Forzando la detención del arreglo II

Ejemplo (continuación) (Recorriendo una lista hasta encontrar un elemento especial)

Un programa que permite solucionar este problema es

```
public static void main(String[] args){
   String[] frutas = new String[]
      {"Pera", "Maracuyá ", "Guayaba", "Lulo", Granadilla"};
   for(String f:frutas) { // para cada elemento f en la lista
      System.out.println(f);
      if( f.equals("Lulo")) {
         break;
      }
   }
}
```


Forzando la detención del arreglo III

Ejemplo (continuación)

El resultado de ejecutar este programa será entonces:

```
...
Pera
Maracuyá
Guayaba
Lulo
...
```

Agenda

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Complementos a los ciclos para (for)
4. Los ciclos para (for) como iteradores de colecciones
5. Problemas

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. Imprimir los números de 1 hasta un número natural n dado, cada uno con su respectivo factorial.
5. Calcular el valor de 2 elevado a la potencia n.
6. Leer un número natural n, leer otro dato de tipo real x y calcular x^n.
7. Diseñe un programa que muestre las tablas de multiplicar del 1 al 9.

Problemas varios II

Problemas

8. Diseñar una función que permita calcular una aproximación de la función exponencial alrededor de 0 para cualquier valor $x \in \mathbb{R}$, utilizando los primeros n términos de la serie de Maclaurin

$$\exp(x, n) \approx \sum_{i=0}^{n} \frac{x^i}{i!}.$$

9. Diseñar una función que permita calcular una aproximación de la función seno alrededor de 0 para cualquier valor $x \in \mathbb{R}$ (x dado en radianes), utilizando los primeros n términos de la serie de Maclaurin

$$\sin(x, n) \approx \sum_{i=0}^{n} \frac{(-1)^i x^{2i+1}}{(2i+1)!}.$$

Problemas varios III

Problemas

10. Diseñar una función que permita calcular una aproximación de la función coseno alrededor de 0 para cualquier valor $x \in \mathbb{R}$ (x dado en radianes), utilizando los primeros n términos de la serie de Maclaurin

$$\cos(x, n) \approx \sum_{i=0}^{n} \frac{(-1)^i x^{2i}}{(2i)!}.$$

11. Diseñar una función que permita calcular una aproximación de la función logaritmo natural alrededor de 0 para cualquier valor $x \in \mathbb{R}^+$, utilizando los primeros n términos de la serie de Maclaurin

$$\ln(x, n) \approx \sum_{i=0}^{n} \frac{1}{2i+1} \left(\frac{x^2 - 1}{x^2 + 1} \right)^{2i+1}.$$

Problemas varios III

Problemas

12. Diseñar una función que permita calcular una aproximación de la función arco tangente para cualquier valor $x \in [-1, 1]$, utilizando los primeros n términos de la serie de Maclaurin (al evaluar esta función el resultado que se obtiene está expresado en radianes)

$$\arctan(x, n) \approx \sum_{i=0}^{n} \frac{(-1)^i x^{2i+1}}{(2i+1)}.$$