5. Esquemas Repetitivos

Objetivos:

- Conocer los tres tipos de esquemas repetitivos más habituales: *mientras, repetir* y *para*
- Identificar las situaciones en las que debe usarse cada uno
- Entender el esquema general y cómo se puede utilizar para simular el resto de esquemas
- Aplicar estos esquemas en ejemplos algorítmicos sencillos
- Conocer los equivalentes en C de los esquemas *mientras, repetir* y *para*

5. Esquemas Repetitivos

1. **Introducción**
2. **El esquema mientras**
3. **El esquema repetir**
4. **El esquema para**
5. **El esquema general**
6. **Equivalencias entre esquemas**
7. **Ejemplos**

Bibliografía:

- Biondi y Clavel.
- Joyanes.
- Kernighan y Ritchie.

5.1. Introducción

- Ya hemos visto que eran necesarios esquemas que repitieran acciones:
- Vamos a estudiar tres tipos de esquemas iterativos: repetir, mientras y para. Además estudiaremos un esquema general que los engloba a todos.

Def. Iteración:
Repetición de una acción o de una secuencia de acciones.

Las condiciones para elegir los esquemas serán:

- ¿ **se conoce a priori el número** de iteraciones?
  - **NO**: utiliza los **esquemas mientras o repetir**;
  - **SI**: utiliza el **esquema para**.
- Si las iteraciones se hacen **0 ó más veces**, utiliza **el esquema mientras**; si se repiten **1 ó más veces** utiliza **el esquema repetir**.

5.2. El esquema mientras

5.2.1. Sintaxis

<esquema_mientras> :: =
mientras <condición> **hacer**
<accion>
fin mientras

Equivalente en C

<esquema_while> :: =
while (<condición>)
<1-accion> |
while (<condicion>**)** {
<acción>
}

5.2.2. Semántica

• Se ejecutan una serie de acciones mientras un predicado (una condición) sea cierto.
• Cuando el predicado sea falso, se termina y se avanza tras el fin mientras.
• No se conoce a priori el número de veces que puede repetirse la secuencia de acciones.
• La acción puede ejecutarse **cero** o más veces(si la condición es falsa inicialmente no se realiza).

5.3. El esquema repetir

5.3.1. Sintaxis

<esquema_repetir> ::=
repetir
<accion>
hasta que <*condición*>

Equivalente en C

<esquema_do_while>::=
do
<1-accion>
while (<condición>)|
do {
<accion>
}**while (**<condicion>**)**

5.3.2. Semántica

• Se repiten una serie de acciones hasta que la condición se cumpla (sea cierta).
• Cuando la condición es cierta, el control sale del esquema iterativo.
• El número de iteraciones es desconocido a priori.
• Las acciones se realizan al menos UNA vez.
• En resumen: *¿qué diferencia hay entre la semántica de los esquemas mientras y repetir?*

5.4. El esquema para

5.4.1. Sintaxis

<Esquema-para> ::=

para <variable> [**de|desde**] **<**expresión-inicial> **[a|hasta]** **<**expresión final> **hacer**
<accion>
Fin para |

para <variable> [**de | desde**] **<**expresión-inicial> **[a | hasta]** **<**expresión final> [**paso + | incremento**] **<**expresión-paso> **hacer**
<accion>
Fin para |

para <variable> [**de | desde**] **<**expresión-inicial> **[a | hasta]** **<**expresión final> [**paso - | decremento**] **<**expresión-paso> **hacer**
<accion>
Fin para

Equivalente en C

<esquema_for> ::=

for (<var = v_ini> ; <expr. fin> ; <expr. incr.>**)**

<1-accion> **|**

for (<var>=<v_ini>; <expr. fin>; <expr. incr.>**) {**

<accion>

}

5.4.2. Semántica

- Se conocen a priori el número de repeticiones
- <*variable*> se conoce como *variable índice*
- Inicialmente se asigna el valor <*expresión inicial*> a la variable índice
- Se comprueba que <variable> no haya sobrepasado el valor de <*expresión final*>
- En cada paso el valor de <variable> aumenta o disminuye en <expresión paso>, o aumenta en 1 en la opción por defecto

5.5. El esquema general

5.1. Sintaxis

<esquema_general>::=

iterar

<acción-1>

salir si <expresión lógica>;

<acción-2>

fin iterar

5.2. Semántica

- Inicialmente se ejecuta <acción-1>.
- Si la <expr. lógica> se evalúa a cierto, se sale del bucle (instrucción siguiente a fin iterar).
- Si la <expr. lógica> se evalúa a falso, se ejecuta <acción-2> y se vuelve a entrar en el bucle.
- <acción-1> se ejecuta al menos una vez, mientras que <acción-2> puede no ejecutarse. Esto es, <acción-1> se ejecuta siempre una vez más que <acción-2>.

5.6. Equivalencias entre esquemas

- Los esquemas **mientras, repetir y para** son particularizaciones del esquema **general.**
- Además, todos los esquemas también pueden realizarse mediante el esquema **mientras** o el esquema **repetir**.

```
mientras p  hacer
  acción_a;
fin mientras
```



```
iterar
  salir si not p;
  acción_a;
fin iterar
```

```
repetir
  acción_b;
hasta que p;
```



```
iterar
  acción_b;
 salir si p;
fin iterar
```

```
Para I desde M hasta N hacer
 acción_c;
Fin para
```



```
I := M;
Iterar
  salir si I > N;
  acción_c;
  I:= I + 1;
Fin iterar
```

```
Mientras p hacer
  a;
fin mientras
```



```
si p entonces
  Repetir
    a;
  hasta que NOT p;
finsi
```

```
para var desde val_inicio
    hasta val_fin paso + inc
    hacer
  a;
fin para
```



```
var := val_inicio;
Mientras var <= val_fin hacer
  a;
  var := var + inc;
fin mientras
```

```
iterar
  a;
  salir si p;
  b;
fin iterar
```



```
a;
Mientras NOT p hacer
  b;
  a;
fin mientras
```

5.7. Ejemplos

Realiza los siguientes ejercicios en código algorítmico (incluyendo P.B.R.) y a continuación tradúcelos a C.

- **Cálculo de la división entera, segura.**
  - Dados dos números enteros, se comprueba si se puede hacer la división y se realiza la división sin utilizar el operador (/ ó div); deben usar restas o sumas sucesivas.

  Algoritmo division_segura (dividendo, divisor, cociente, resto, ok) es

  Dividendo, dividor: numérico; {parámetro dato}

  Cociente, resto: numérico; {parámetro resultado; cociente y resto de la división entera}

  Ok:lógico; {parámetro resultado; cierto si se pudo hacer}
- **Comprobar si un número es primo.**
  - Dado un número n, responde cierto/falso sobre si el número es primo o no.

  Algoritmo es_primo (n, resp) es

  N: numérico; {parámetro dato}

  Resp: lógico; {parámetro resultado}

5.7. Ejemplos (cont.)

- **Cálculo de la raíz entera de un número**.
  - Dado un número n, calcula el valor r, entero, que se aproxima por abajo a su raíz cuadrada real.

  Algoritmo raiz_entera (n, raiz) es

  N: numérico; {parámetro dato}

  raiz: numérico; {entero; parámetro resultado}

- **Elección de una opción en un menú.**
  - Se ofrecen distintas opciones al usuario, que debe seleccionar una opción válida.

  Algoritmo menu (opción) es

  opción: numérico; {entero; parámetro resultado}

- **Suma de los N primeros números naturales**
  - Dado un número n, calcula la suma de los números entre 1 y n.

  Algoritmo suma_naturales (n, suma) es

  N: numérico; {parámetro dato}

  suma: numérico; {entero; parámetro resultado}

5.7. Ejemplos (cont.)

- **Mostrar por pantalla los N primeros términos de la sucesión de Fibonacci.**
  - Dado un número entero positivo, N, mostrar por pantalla los N primeros términos de la sucesión de Fibonacci.

  Algoritmo fibonacci (n) es

  N: numérico; {parámetro dato}
- **Suma de los N primeros términos de una progresión geométrica.**
  - Dado un número n, calcula la suma de los n primeros términos de la sucesión con inicio *ni* y razón *r*.

  Algoritmo suma_progresion (n, suma, ni, r) es

  n, ni, r: numérico; {parámetro dato}

  suma: numérico; {parámetro resultado}

Realiza ahora una solución equivalente de los mismos ejercicios utilizando el esquema *iterar*