Programación 1

Tema 4

Instrucciones simples y estructuradas

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Información sobre protección de datos de carácter personal en el tratamiento de gestión de grabaciones de docencia

Sesión con grabación

Tratamiento: Gestión de grabaciones de docencia

Finalidad: Grabación y tratamiento audiovisual de docencia y su evaluación

Base Jurídica: Art. 6.1.b), c) y d) Reglamento General de Protección de Datos

Responsable: Universidad de Zaragoza.

Ejercicio de Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento ante el gerente de la Universidad conforme a https://protecciondatos.unizar.es/procedimiento-seguir

Información completa en:

https://protecciondatos.unizar.es/sites/protecciondatos.unizar.es/files/users/lopd/gdocencia_extensa.pdf

Propiedad intelectual: Queda prohibida la difusión, distribución o divulgación de la grabación y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes. La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa y de índole civil o penal.

Fuente de las imágenes: https://pixabay.com/es

Información sobre protección de datos de carácter personal en el tratamiento de gestión de grabaciones de docencia

- Se recuerda que la grabación de las clases por medios distintos a los usados por el profesor o por personas diferentes al profesor sin su autorización expresa no está permitida, al igual que la difusión de esas imágenes o audios.

Índice

- Instrucciones simples
- Instrucciones estructuradas

Instrucción

```
<instrucción> ::=
  <instrucción-simple> |
  <instrucción-estructurada>
```

Instrucciones. Instrucciones simples

- Instrucciones simples
  - De declaración
  - De expresión
    - Asignación
    - Incremento y decremento
    - Entrada y salida
    - Instrucción nula
  - De invocación
  - De devolución de valor

Instrucciones. Instrucciones estructuradas

- Instrucciones estructuradas
  - Bloques secuenciales de instrucciones
  - Instrucciones condicionales
  - Instrucciones iterativas
    - Bucles *while*
    - Instrucciones iterativas indexadas (bucles *for*)

Instrucciones simples de declaración

```
int x;
int i, j, k;
bool b;
int a = 100;
char c1 = 'h';
bool b = true;
double r2 = 1.5e6;
int n = 4 + 8;
char c = char(int('A') + 1);
boolean esDoce = (n == 12);
double r = sqrt(2.0);
const double PI = 3.141592653589793;
```

Instrucciones simples de expresión

- **Asignación**
  - `x = 10;`
  - `x = x + 10;`
  - `x += 10;`
- **Incremento** y **decremento**
  - `x++;`
  - `x--;`
- **Entrada** y **salida**
  -
```
cout << "Resultado: " << fixed
     << setprecision(2) << setw(8)
     << 2 * PI * r << endl;
```
  - `cin >> n1 >> n2;`

Instrucciones simples

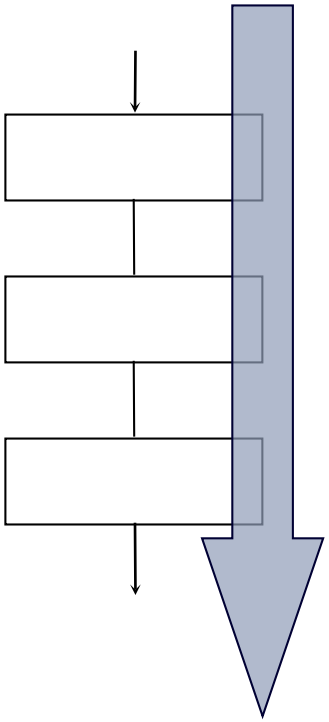
- Nula
  - `;`
- Invocación a un procedimiento
  - `presentarTabla(7);`
- Devolución de valor en una función
  - `return 0;`
  - `return n;`
  - `return 2 * PI * r;`

Instrucciones estructuradas

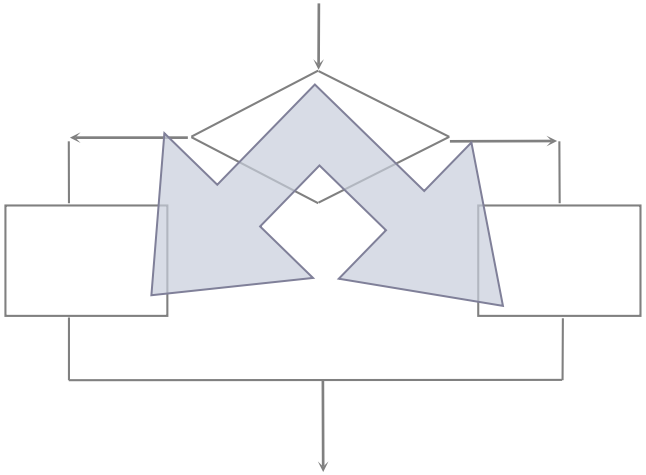
```
<instrucción-estructurada> ::=
    <bloque-secuencial> |
    <instrucción-condicional> |
    <instrucción-iterativa> |
    <instrucción-iterativa-indexada>
```

Instrucciones estructuradas

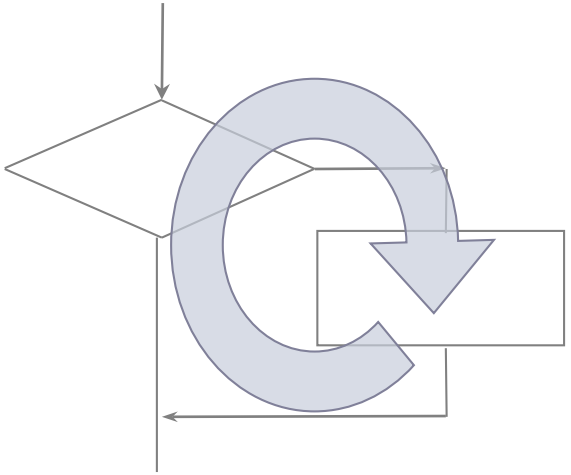
Secuencial



Condicional



Iterativa



Composición secuencial

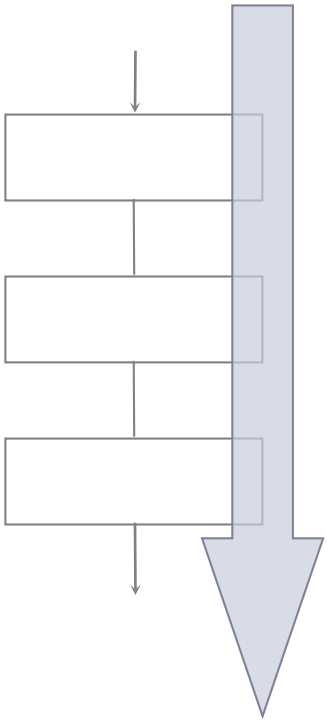
<bloque-secuencial> ::=
“{” { <instrucción> } “}”

Ejemplo de composición secuencial

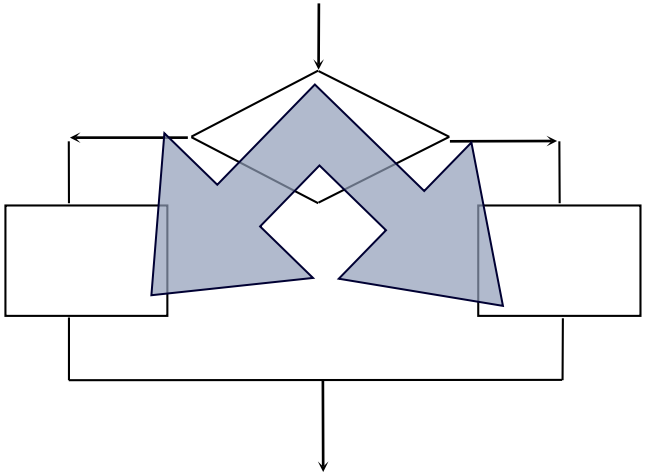
```
int main() {
   // Definición de la constante de cambio
   const double PTAS_POR_EURO = 166.386;

   // Petición y lectura del valor de pesetas
   cout << "Escriba una cantidad en pesetas: ";
   int pesetas;
   cin >> pesetas;

   // Cálculo del equivalente en euros
   double euros = pesetas / PTAS_POR_EURO;

   // Escritura de resultados
   cout << fixed << setprecision(2) << euros << endl;

   return 0;
}
```

Instrucciones estructuradas

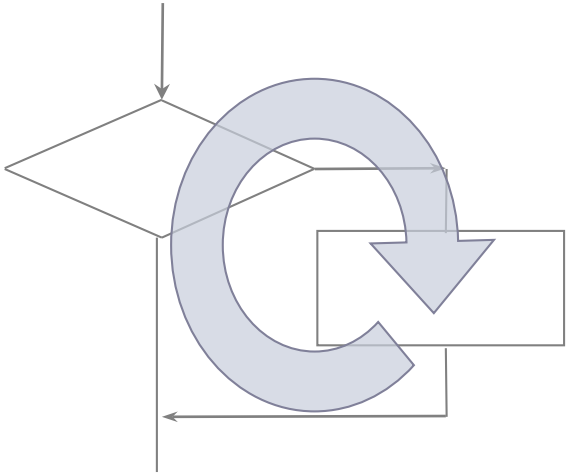
Secuencial



Condicional



Iterativa



Composición condicional

```
<instrucción-condicional> ::=
  “if” “(” <condición> “)”
    <instrucción>
    [“else” <instrucción>]

<condición> ::= <expresión>
```

Composición condicional

- Semántica
  - Se evalúa la condición.
  - Si el valor resultante es *cierto*, se ejecuta únicamente la instrucción que sigue a la condición, una sola vez.
  - Si el valor resultante es *falso* y hay una cláusula **`else`**, se ejecuta únicamente la instrucción de la cláusula **`else`**, una sola vez.

Composición condicional

```
if (x >= 0) {
  cout << x << endl;
}
else {
  cout << -x << endl;
}
```

Programa completo

```
#include <iostream>
using namespace std;

/*
 * Programa que pide al usuario un número y escribe en la pantalla el valor
 * absoluto de este.
 */
int main() {
    cout << "Introduzca un número: ";
    double x;
    cin >> x;

    cout << "Su valor absoluto es: ";
    if (x >= 0.0) {
        cout << x << endl;
    }
    else {
        cout << -x << endl;
    }

    return 0;
}
```

Otro ejemplo

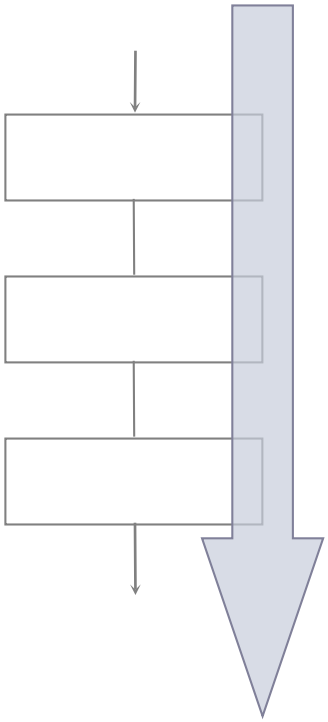
- Trozo de código que ordene los valores de dos variables enteras a y b, de forma que $a \leq b$

Otro ejemplo

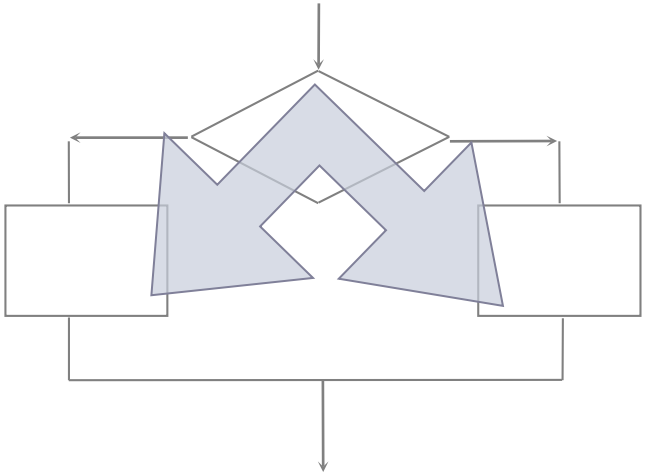
```
// Ordena los valores de las variables a y b

int a = ...;
int b = ...;

// a = A0 y b = B0
if (a > b) {
    int temp = a;
    a = b;
    b = temp;
}
// a ≤ b y ((a = A0 y b = B0) o (a = B0 y b = A0))
```

Instrucciones estructuradas

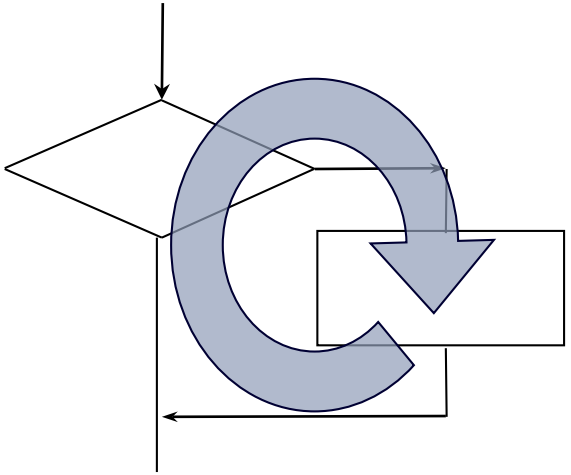
Secuencial



Condicional



Iterativa



Ejemplo. Programa que escriba en la pantalla una tabla de multiplicar

```
Introduzca un número: 7

LA TABLA DEL 7
  7 x  0 =   0
  7 x  1 =   7
  7 x  2 =  14
  7 x  3 =  21
  7 x  4 =  28
  7 x  5 =  35
  7 x  6 =  42
  7 x  7 =  49
  7 x  8 =  56
  7 x  9 =  63
  7 x 10 =  70
```

Ejemplo. Programa que escriba en la pantalla una tabla de multiplicar

```
/*
 * Programa que solicita un número entero al usuario y
 * escribe en la pantalla la tabla de multiplicar
 * correspondiente a ese número.
 */
int main() {
    cout << "Introduzca un número: ";
    int n;
    cin >> n;

    // Escribe la cabecera de la tabla
    cout << endl;
    cout << "LA TABLA DEL " << n << endl;

    // Escribe las 11 líneas de la tabla
    ...

    return 0;
}
```

¿Una solución?

```
cout << setw(3) << n << " x   0 =   0"                    << endl;
cout << setw(3) << n << " x   1 = " << setw(3) << n       << endl;
cout << setw(3) << n << " x   2 = " << setw(3) << n *  2 << endl;
cout << setw(3) << n << " x   3 = " << setw(3) << n *  3 << endl;
cout << setw(3) << n << " x   4 = " << setw(3) << n *  4 << endl;
cout << setw(3) << n << " x   5 = " << setw(3) << n *  5 << endl;
cout << setw(3) << n << " x   6 = " << setw(3) << n *  6 << endl;
cout << setw(3) << n << " x   7 = " << setw(3) << n *  7 << endl;
cout << setw(3) << n << " x   8 = " << setw(3) << n *  8 << endl;
cout << setw(3) << n << " x   9 = " << setw(3) << n *  9 << endl;
cout << setw(3) << n << " x  10 = " << setw(3) << n * 10 << endl;
```

Composición iterativa

```
<instrucción-iterativa> ::=
    “while” “(” <condición> “)”
        <instrucción>
```

Composición iterativa

- Semántica
  - Se evalúa la `condición`
    → resultado *cierto* o *falso*
  - Mientras se evalúa como *cierto*, se ejecuta la `instrucción` que sigue a la `condición` y se vuelve a evaluar la `condición`
  - Cuando se evalúa como *falso*, concluye la ejecución de la instrucción iterativa

Ejemplo

```
// Escribe las 11 líneas de la
// tabla de multiplicar del «n»
unsigned int i = 0;
while (i <= 10) {
    cout << setw(3) << n
         << " x "
         << setw(2) << i
         << " = "
         << setw(3) << n * i
         << endl;
    i = i + 1;
}
```

Un problema

- Programa que calcule el factorial de un número

```
Escriba un número natural: 5
5! = 120
```

Factorial

```
#include <iostream>
using namespace std;

/*
 * Programa que pide al usuario un número natural, lo lee del teclado y
 * escribe en la pantalla su factorial.
 */
int main() {
    cout << "Escriba un número natural: ";
    unsigned int n;
    cin >> n;

    // Asigna a «factorial» el valor de «n»!, siendo n>=0
    unsigned int factoral = ...;
    ...

    cout << n << "! = " << factorial << endl;
    return 0;
}
```

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;       // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;     // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;       // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;     // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n 4

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;       // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;     // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	1

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	1
fact	1

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

true

n	4
i	1
fact	1

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	1
fact	1

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	2
fact	1

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	2
fact	1

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	2
fact	2

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

true

n	4
i	2
fact	2

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	2
fact	2

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	3
fact	2

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	3
fact	2

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;       // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;     // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	3
fact	6

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

true

n	4
i	3
fact	6

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	3
fact	6

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	4
fact	6

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;       // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;     // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	4
fact	6

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;       // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;     // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

n	4
i	4
fact	24

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
```

false

n	4
i	4
fact	24

Factorial

```
/*
 * Asigna a «factorial» el valor de n!,
 * siendo n>=0
 */
unsigned int i = 1;
unsigned int factorial = 1;      // factorial = i!
while (i < n) {
   i++;
   factorial = i * factorial;    // factorial = i!
}
// i = n, factorial = i! → factorial = n!
...
```

n	4
i	4
fact	24

Factorial

- Otra versión del código, utilizando una función
  - Código mejor estructurado
  - Más reutilizable

… más detalles en el tema 5

Factorial

```
#include <iostream>
using namespace std;

/*
 * Devuelve n!
 */
unsigned int factorial(unsigned int n) {
   ...
}

/*
 * Programa que pide al usuario un número natural, lo lee del
 * teclado y escribe en la pantalla su factorial.
 */
int main() {
   ...
}
```

Factorial

```
/*
 * Programa que pide al usuario un número natural, lo lee
 * del teclado y escribe en la pantalla su factorial.
 */
int main() {
   cout << "Escriba un número natural: ";
   unsigned int n;
   cin >> n;

   cout << n << "! = " << factorial(n) << endl;

   return 0;
}
```

Factorial

```
/*
 * Devuelve n!
 */
unsigned int factorial(unsigned int n) {
   // Asigna a «factorial» el valor de «n»!, siendo n>=0
   unsigned int i = 1;
   unsigned int factorial = 1;          // factorial = i!
   while (i < n) {
      i++;
      factorial = i * factorial; // factorial = i!
   }
   // i = n, factorial = i! ==> factorial = n!
   return factorial;
}
```

Composición iterativa indexada

```
<instrucción-iterativa-indexada> ::=
  “for” “(” <inicialización> “;”
            <condición> “;”
            <actualización> “)”
    <instrucción>

<inicialización> ::= <instrucción>
<condición> ::= <expresión>
<actualización> ::= <instrucción>
```

Composición iterativa indexada

- Semántica
  - Se ejecuta la instrucción de inicialización
  - Se evalúa la condición → resultado *cierto* o *falso*
  - Mientras el resultado es *cierto*:
    - Se ejecuta la instrucción del cuerpo del bucle
    - Se ejecuta la instrucción de actualización
    - Se vuelve a evaluar la condición
  - Cuando el resultado el *falso*, concluye la ejecución de la instrucción iterativa indexada

Ejemplo

```
void presentarTabla(int n) {
   // Escribe la cabecera de la tabla de multiplicar del «n»
   cout << endl;
   cout << "LA TABLA DEL " << n << endl;

   // Escribe las 11 líneas de la tabla de multiplicar del «n»
   for (unsigned int i = 0; i <= 10; i++) {
      cout << setw(3) << n
           << " x " << setw(2) << i
           << " = " << setw(3) << n * i
           << endl;
   }
}
```

Equivalencias bucles *while* y *for*

```
for ( <inicialización>; <condición>;
            <actualización> )
   <instrucción>


<inicialización>;
while (<condición>) {
  <instrucción>;
  <actualización>;
}
```

Factorial

```
/*
 *  Asigna a «factorial» el valor n!, con n>=0
 */
unsigned int factorial = 1;
for (unsigned int i = 1; i <= n; i++) {
   factorial = i * factorial;  // factorial = i!
}
// factorial = n!
```

Índice

- Instrucciones simples
- Instrucciones estructuradas

¿Cómo se puede estudiar este tema?

- Repasando estas transparencias
- Trabajando con el código de estas transparencias
  - https://github.com/prog1-eina/tema-04-instrucciones
- Leyendo el capítulo 5 de los apuntes del profesor Martínez
  - Disponible en Moodle
- Realizando algunos de los ejercicios básicos sobre instrucción condicional e iterativa disponibles en Moodle:
  - https://moodle.unizar.es/add/mod/page/view.php?id=1872834
  - https://moodle.unizar.es/add/mod/page/view.php?id=1872835