Programación 1

Tema 5

Funciones

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Índice

- **Funciones**
- Especificación de funciones
- Estructuración de programas
  - Metodología de programación descendente
- Ámbito y vida
- Comunicación entre funciones

Funciones

- Abstracción de un conjunto de instrucciones
  - a las que se la un nombre determinado
  - para ser invocadas desde algún otro punto del programa

Fuente: «Functions». *Cplusplus.com.* 2000–2017. http://www.cplusplus.com/doc/tutorial/functions/ (consultado el 8-10-2020)

Funciones

- Pretenden ser un reflejo del concepto matemático de *función*:
  - Relación binaria entre dos conjuntos que asocia a cada elemento del primer conjunto exactamente un elemento del segundo conjunto.
  - $f: \mathbb{R} \longrightarrow \mathbb{R}$
  - $x \mapsto f(x)$

Fuente: Wikipedia contributors. «Function (mathematics)». *Wikipedia, The Free Encyclopedia.* 2020. https://en.wikipedia.org/w/index.php?title=Function_(mathematics)&oldid=981590415 (consultado el 8-10-2020)

Funciones
Ejemplo matemático

- Ejemplo:
  - $f: \mathbb{R} \longrightarrow \mathbb{R}$
  - $x \mapsto x^2$

Funciones
Ejemplo en C++

```
/*
 * Devuelve el cuadrado de x
 */
double f(double x) {
    return x * x;
}
```

Funciones
Ejemplo de definición

```
/*
 * Devuelve el cuadrado de x
 */
double cuadrado(double x) {
    return x * x;
}
```

Funciones en C++

- Sintaxis:
  - Declaración
  - Definición
  - Invocación

Más adelante en el curso, no ahora

Funciones. Sintaxis

```
<definición-función> ::=
    <tipo> <identificador>
        “(” [<lista-parámetros>] “)”
    <bloque-secuencial>
<lista-parámetros> ::=
    <parámetro> { “,” <parámetro> }
<parámetro> ::=
    <tipo> <identificador>
```

Funciones

Ejemplo en C++

```
/*
 * Devuelve el cuadrado de x
 */
double cuadrado(double x) {
    return x * x;
}
```

Funciones. Sintaxis

```
<invocación-función> ::=
    <identificador>
    “(” [ <lista-argumentos> ] “)”
<lista-argumentos> ::=
    <argumento> {“,” <argumento>}
<argumento> ::= <expresión>
```

Funciones

Ejemplos de invocaciones

```
cuadrado(2)

cuadrado(x)

cuadrado(2 + x)

cuadrado(cuadrado(sqrt(x))
```

Funciones

Otro ejemplo

esBisiesto: $\mathbb{N} \longrightarrow \mathbb{B}$

$$n \mapsto \begin{cases} \text{true} & \text{si } n \text{ representa un año bisiesto} \\ \text{false} & \text{si } n \text{ representa un año no bisiesto} \end{cases}$$

Funciones
Otro ejemplo

```
/* Declaración */
bool esBisiesto(unsigned int agno) {
    bool multiplo4   = (agno %   4 == 0);
    bool multiplo100 = (agno % 100 == 0);
    bool multiplo400 = (agno % 400 == 0);
    return multiplo400 || (multiplo4 && !multiplo100);
}
```

```
/* Invocaciones */
esBisiesto(2020)
esBisiesto(agno)
esBisiesto(agno + 1)
esBisiesto(agno + 4)
```

Procedimientos

Funciones que no devuelven valor

```
void saludar() {
    cout << "¡Hola, mundo!" << endl;
}
```

Funciones. Sintaxis

- Restricciones a la sintaxis:
  - Si el tipo devuelto es distinto de **`void`**, el cuerpo de la función debe devolver un dato del tipo adecuado a través de la instrucción **`return`**.
  - El identificador de la invocación es el mismo que el de la definición.
  - La lista de parámetros (definición) y la de argumentos (invocación) tienen el mismo número de elementos.
  - El tipo del *i*-ésimo argumento en la lista de argumentos es el mismo (o es compatible) con el *i*-ésimo parámetro de la definición.

Funciones

Otro ejemplo de definición

```
/*
 * Dado un polígono regular con un número de lados
 * igual al valor del parámetro «numLados» de
 * longitud igual al valor del parámetro «longitud»,
 * devuelve el perímetro de dicho polígono regular.
 * «numLados» tiene que ser mayor o igual que 3 y
 * «longitud» mayor que 0.0.
 */
double perimetro(unsigned int numLados,
                 double longitud) {
    return numLados * longitud;
}
```

Funciones

Otros ejemplos de invocaciones

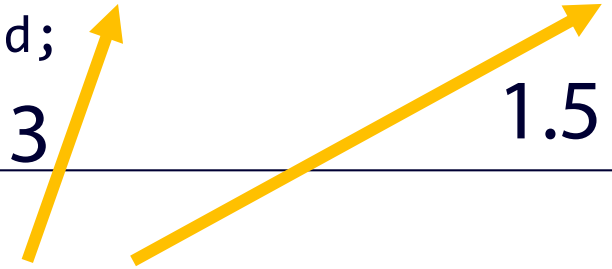
```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```

```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```

Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```




```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```

Funciones

Otros ejemplos de invocaciones

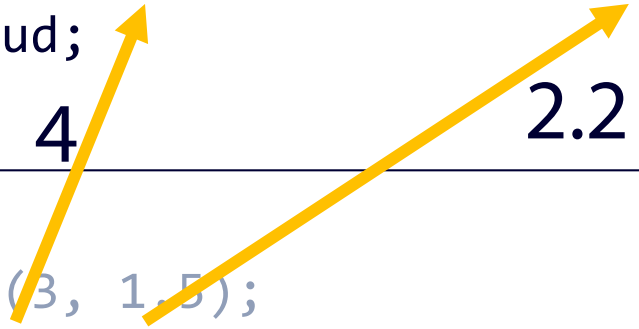
```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```


```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```

Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```




```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```

Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```

8.8

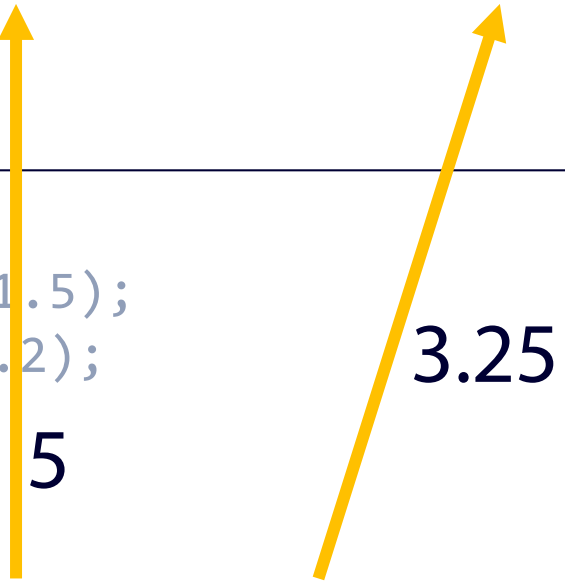
```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```

Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```

```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```

16.25

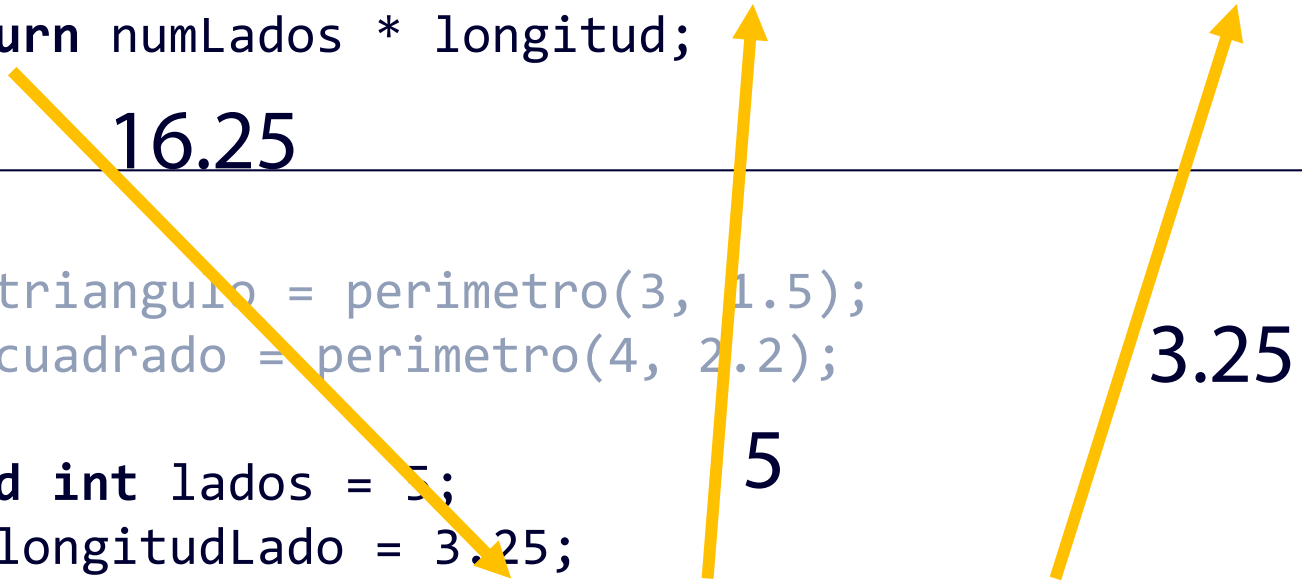
```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```

Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```

```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int lados = 5;
double longitudLado = 3.25;
double pentagono = perimetro(lados, longitudLado);

cout << perimetro(lados + 1, longitudLado - 1) << endl;
```

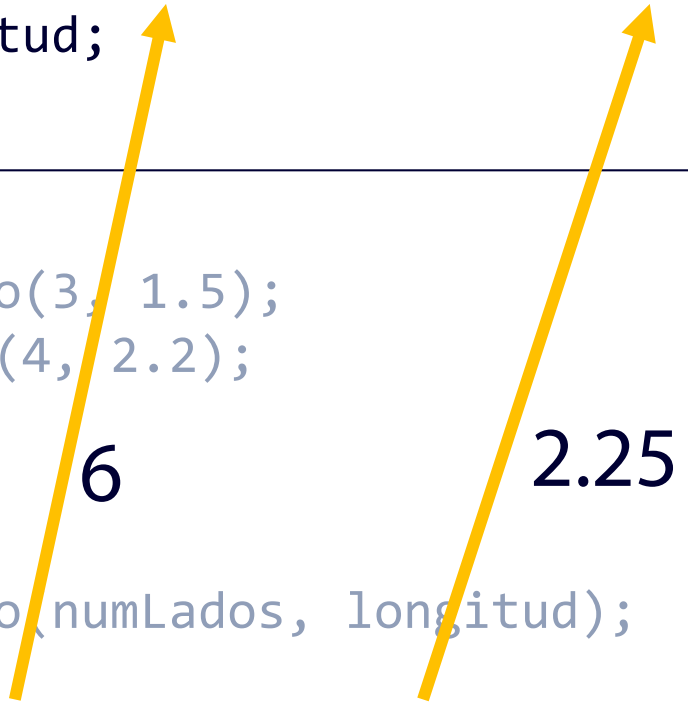


Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```

```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Funciones

Otros ejemplos de invocaciones

```
double perimetro(unsigned int numLados, double longitud) {
    return numLados * longitud;
}
```

13.5

```
...
double triangulo = perimetro(3, 1.5);
double cuadrado = perimetro(4, 2.2);

unsigned int numLados = 5;
double longitud = 3.25;
double pentagono = perimetro(numLados, longitud);

cout << perimetro(numLados + 1, longitud - 1) << endl;
```

Índice

- Funciones
- **Especificación de funciones**
- Estructuración de programas
  - Metodología de programación descendente
- Ámbito y vida
- Comunicación entre funciones

Especificación de funciones

```
/*
 * Pre:  P
 * Post: Q
 */
void f() {
   ...
}
```

Si se cumple la precondición *P* inmediatamente antes de invocar a la función f, entonces f se ejecuta, termina y se alcanza un estado en el que se cumple la postcondición *Q*.

Especificación de funciones

```
/*
 * Pre:  ---
 * Post: Ha devuelto el valor del polinomio
 *       ax² + bx + c
 */
double calcular(double a, double b, double c,
                double x) {
  return ((a * x + b) * x) + c;
}
```

Especificación de funciones

```
/*
 * Pre:  1 ≤ dia ≤ 31, 1 ≤ mes ≤ 12, anyo > 0
 * Post: Ha escrito en la pantalla una línea con
 *       la fecha definida por los valores de
 *       los parámetros «dia», «mes» y «anyo»
 *       con el siguiente formato: dia/mes/anyo.
 *       Por ejemplo: 12/1/2014
*/
void escribirFecha(int dia, int mes, int anyo) {
   cout << dia << "/" << mes << "/" << anyo << endl;
}
```

Especificación de funciones

```
/*
 * Pre:  ---
 * Post: Ha presentado por pantalla una línea
 *       con el texto “En esta asignatura se
 *       aprende a programar”
 */
void anunciar() {
   cout <<
      "En esta asignatura se aprende a programar"
      << endl;
}
```

Especificación de funciones

```
/*
 * Pre:  n >= 0
 * Post: Ha devuelto el valor de n!
 */
int factorial(int n) {
    ...
}
```

Índice

- Funciones
- Especificación de funciones
- **Estructuración de programas**
  - **Metodología de programación descendente**
- Ámbito y vida
- Comunicación entre funciones

```
¿Qué tabla desea escribir? (0 para acabar): 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

¿Qué tabla desea escribir? (0 para acabar): 6

LA TABLA DEL 6

6 x  0 =   0
6 x  1 =   6
6 x  2 =  12
6 x  3 =  18
6 x  4 =  24
6 x  5 =  30
6 x  6 =  36
6 x  7 =  42
6 x  8 =  48
6 x  9 =  54
6 x 10 =  60

¿Qué tabla desea escribir? (0 para acabar): 0
```

Una (mala) solución

```
...
/* Programa que pregunta reiteradamente al usuario qué tabla … */
int main() {
    cout << "¿Qué tabla desea escribir (0 para acabar)?: ";
    int multiplicando;
    cin  >>  multiplicando;
    while (multiplicando != 0) {
        cout << endl;
        cout << "LA TABLA DEL " << multiplicando << endl;
        for (int i = 0; i <= 10; ++i) {
            cout << setw(3) << multiplicando << " x " << setw(2) << i
                 << " = " << setw(3) << multiplicando * i << endl;
        }
        cout << endl << "¿Qué tabla desea escribir (0 para acabar)?: ";
        cin >> multiplicando;
   }
   return 0;
}
```

```
¿Qué tabla desea escribir? (0 para acabar): 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

¿Qué tabla desea escribir? (0 para acabar): 6
LA TABLA DEL 6

6 x  0 =   0
6 x  1 =   6
6 x  2 =  12
6 x  3 =  18
6 x  4 =  24
6 x  5 =  30
6 x  6 =  36
6 x  7 =  42
6 x  8 =  48
6 x  9 =  54
6 x 10 =  60

¿Qué tabla desea escribir? (0 para acabar): 0
```

Función `main`

```
/*
 * Programa que pregunta reiteradamente al usuario qué
 * tabla de multiplicar desea escribir y la escribe
 * a continuación, salvo cuando el usuario responde con un
 * 0, en cuyo caso el programa termina.
*/
int main() {
   …
}
```

Función main

```
int main() {
   cout << "Tabla que desea escribir (0 acaba): ";
   int multiplicando;
   cin >> multiplicando;

   while (multiplicando != 0) {
      presentarTabla(multiplicando);
      cout << Tabla que desea escribir (0 acaba): ";
      cin >> multiplicando;
   }
   return 0;
}
```

```
¿Qué tabla desea escribir? (0 para acabar): 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

¿Qué tabla desea escribir? (0 para acabar): 6
LA TABLA DEL 6

6 x  0 =   0
6 x  1 =   6
6 x  2 =  12
6 x  3 =  18
6 x  4 =  24
6 x  5 =  30
6 x  6 =  36
6 x  7 =  42
6 x  8 =  48
6 x  9 =  54
6 x 10 =  60

¿Qué tabla desea escribir? (0 para acabar): 0
```

Función presentarTabla

```
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
```

Función presentarTabla

```
/*
 *  Presenta en la pantalla la tabla de multiplicar
 *  del «n»:
 *
 *         LA TABLA DEL «n»
 *         «n» x  0 =  0
 *         «n» x  1 = «n»
 *         «n» x  2 = ...
 *              ...
 *         «n» x  9 = ...
 *         «n» x 10 = ...
 */
void presentarTabla(int n) {
   …
}
```

Función presentarTabla

```
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 (int i = 0; i <= 10; i++) {
      cout << setw(3) << n
           << " x "
           << setw(2) << i
           << " = "
           << setw(3) << n * i
           << endl;
   }
}
```

Esquema programa completo

```
/*
 * Autores: Miguel Ángel Latre y Javier Martínez
 * Última revisión: 8 de octubre de 2020
 * Resumen: Programa interactivo que presenta por pantalla las tablas de
 *          multiplicar seleccionadas por el usuario
 */
#include <iostream>
#include <iomanip>
using namespace std;

/* … */
void presentarTabla(int n) {
    …
}

/* … */
int main() {
    …
}
```

Índice

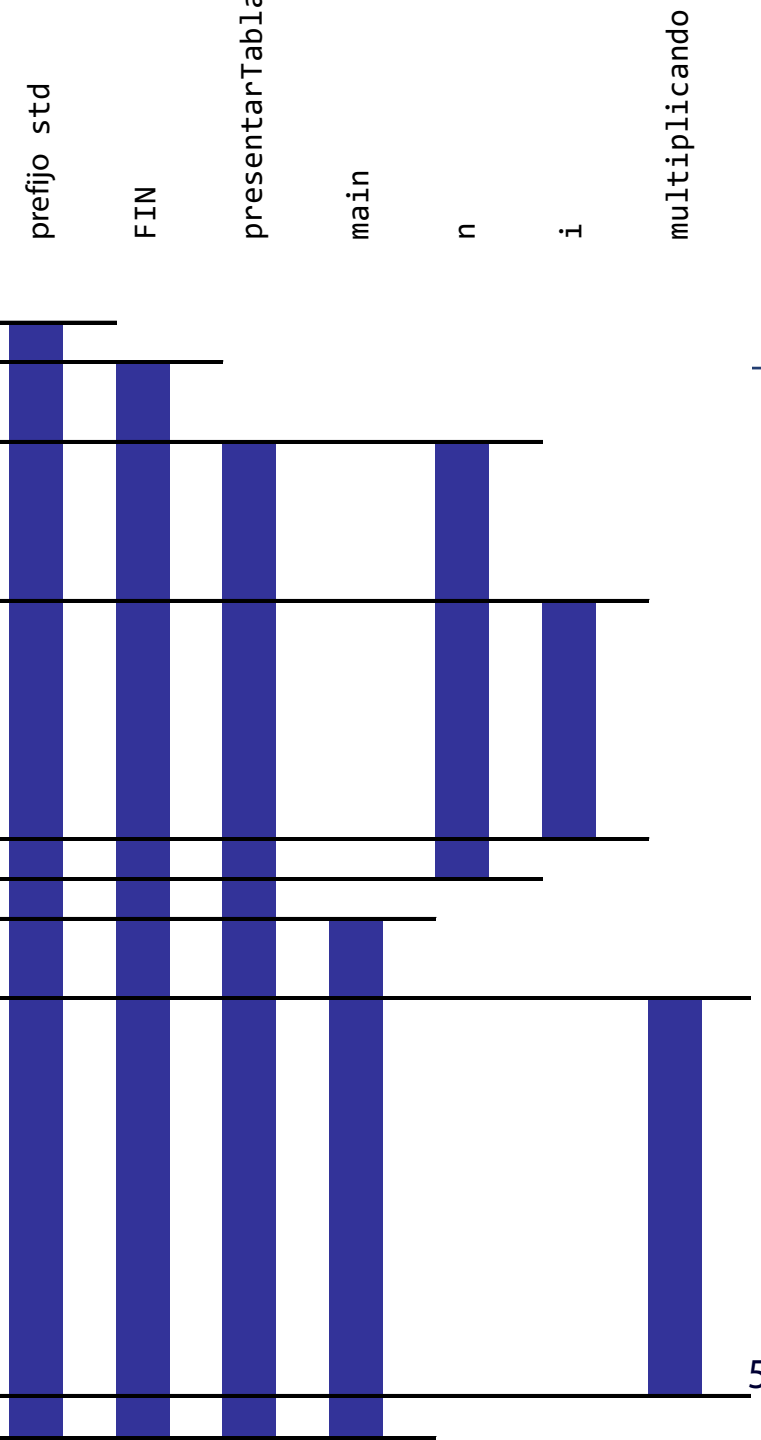
- Funciones
- Especificación de funciones
- Estructuración de programas
  - Metodología de programación descendente
- **Ámbito y vida**
- Comunicación entre funciones

Ámbito y vida

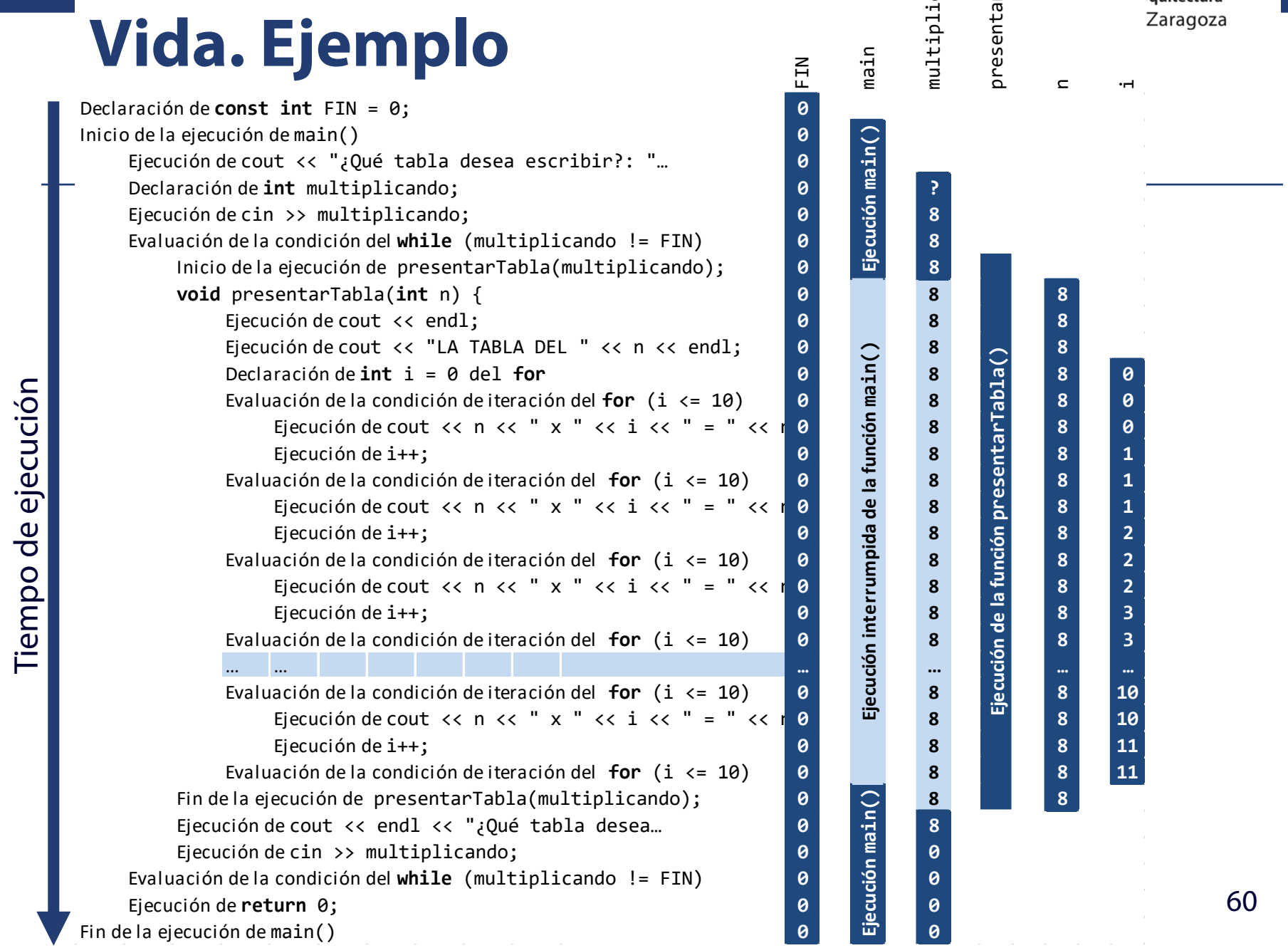
- Elemento nombrado con un identificador: función, constante, variable, parámetro, …
- **Ámbito** o **visibilidad** (*scope*): zona del código en la que un elemento es accesible (se puede hacer uso de él).
  - **Ámbito local** de los elementos definidos dentro de un bloque o función:
    - Desde el punto en que se definen hasta el final del bloque o función.
  - **Ámbito global** de los elementos definidos en el fichero fuera de las funciones:
    - Desde el punto en que se han definido hasta el final del fichero.
- **Duración** o **vida** (*lifetime*) de un elemento
  - Tiempo en el que el elemento está disponible durante la ejecución del programa.

Ámbito. Ejemplo

```
#include <iostream>
#include <iomanip>
using namespace std;
const int FIN = 0;

void presentarTabla(int n) {
     cout << endl;
     cout << "LA TABLA DEL " << n << endl;

     for (int i = 0; i <= 10; i++) {
          cout << setw(3) << n
               << " x " << setw(2) << i
               << " = " << setw(3) << n * i
               << endl;
     }
}

int main() {
     cout << "¿Qué tabla desea escribir?: "…
     int multiplicando;
     cin  >>  multiplicando;
     while (multiplicando != FIN) {
          presentarTabla(multiplicando);
          cout << endl << "¿Qué tabla desea…
          cin >> multiplicando;
     }

     return 0;
}
```



Vida. Ejemplo



Índice

- Funciones
- Especificación de funciones
- Estructuración de programas
  - Metodología de programación descendente
- Ámbito y vida
- **Comunicación entre funciones**

Comunicación entre funciones

- Parámetros por valor — Ya vistos
- Parámetros por referencia
- Valor devuelto — Ya visto
- Variables globales

Otro problema distinto

```
int a, b;
...
// Si ahora: a = X y b = Y …
intercambiar(a, b);
// … entonces ahora: a = Y y b = X
```

Comunicación por valor. Solución errónea

```
/*
 * Pre:  uno = X0 y otro = Y
 * Post: uno = Y0 y otro = X0
 */
void intercambiar(int uno, int otro) {
  int aux = uno;
  uno = otro;
  otro = aux;
}
```

Comunicación por valor. Solución errónea

```
#include <iostream>
using namespace std;
void intercambiar(int uno, int otro) {
    int aux = uno;
    uno = otro;
    otro = aux;
}
int main() {
    int a = 20;
    int b = 4;
    intercambiar(a, b);
    cout << a << " " << b << endl;
    return 0;
}
```

Comunicación por valor. Solución errónea

- Ejecución en C++ Tutor

Comunicación mediante parámetros por referencia

```
/*
 * Pre:  uno = X0 y otro = Y0
 * Post: uno = Y0 y otro = X0
 */
void intercambiar(int& uno, int& otro) {
  int aux = uno;
  uno = otro;
  otro = aux;
}
```

Comunicación mediante parámetros por referencia

- Ejecución en C++ Tutor

Comunicación mediante parámetros por referencia

```
/*
 * Pre:  a = X y b = Y
 * Post: «a» almacena el menor de los
 *       valores {X, Y} y «b» almacena el
 *       mayor de los valores {X, Y}.
 */
void ordenar(int& a, int& b) {
  if (a > b) {
    intercambiar(a, b);
  }
}
```

Comunicación mediante parámetros por referencia

```
/*
 * Pre:  a = X, b = Y y c = Z
 * Post: «a» almacena el menor de los valores
 *       {X, Y, Z}, «c» almacena el mayor de
 *       los valores {X, Y, Z} y «b» almacena el
 *       valor intermedio de {X, Y, Z}.
 */
void ordenar(int& a, int& b, int& c) {
   ordenar(a, b);
   ordenar(b, c);
   ordenar(a, b);
}
```

Comunicación mediante parámetros por referencia

```
/*
 * Pre:  ---
 * Post: Ha asignado a los parámetros «nacimiento», «estatura» y «peso» los
 *       valores determinados por el usuario como respuesta a tres preguntas
 *       que le son formuladas acerca de su año de nacimiento, su
 *       estatura y su peso.
*/
void preguntarDatos(int& nacimiento, double& estatura, double& peso) {
    cout  << "Escriba año de nacimiento: ";          // 1.ª pregunta
    cin >> nacimiento;                               // lee la respuesta
    cout  << "Su estatura: ";                        // 2.ª pregunta
    cin >> estatura;                                 // lee la respuesta
    cout  << "Su peso: ";                            // 3.ª pregunta
    cin >> peso;                                     // lee la respuesta
}
```

Comunicación entre funciones

- Parámetros por valor
- Parámetros por referencia
- Valor devuelto
- Variables globales
  - **¡PROHIBIDAS EN ESTE CURSO!**

Problemas del uso de variables globales

- Diseño dependiente del nombre de las variables globales
  - Reducción de la capacidad de reutilización del código
- Efectos laterales debidos a la posibilidad de modificación de sus valores desde cualquier parte del código
- Reducción de la legibilidad de las funciones

Índice

- Funciones
- Especificación de funciones
- Estructuración de programas
  - Metodología de programación descendente
- Ámbito y vida
- Comunicación entre funciones

¿Cómo se puede estudiar este tema?

- Repasando estas transparencias
- Trabajando con el código de estas transparencias
  - https://github.com/prog1-eina/tema-05-funciones
- Leyendo «Functions». *Cplusplus.com*. 2000–2017 http://www.cplusplus.com/doc/tutorial/functions/
- Leyendo el capítulo 4 de los apuntes del profesor Martínez
  - Disponible en Moodle
- Realizando algunos de los ejercicios básicos sobre funciones disponibles en Moodle:
  - https://moodle.unizar.es/add/mod/page/view.php?id=1528499