Fundamentos de la programación

4 La abstracción procedimental

Grado en Ingeniería Informática
Grado en Ingeniería del Software
Grado en Ingeniería de Computadores

Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense

Índice

Luis Hernández Yáñez

Diseño descendente
Tareas y subtareas

Luis Hernández Yáñez

Tareas y subtareas

Refinamientos sucesivos

Tareas que ha de realizar un programa:

Se pueden dividir en subtareas más sencillas

Subtareas:

También se pueden dividir en otras más sencillas...

→ Refinamientos sucesivos

Diseño en sucesivos pasos en los se amplía el detalle

Ejemplos:

- ✓ Dibujar 
- ✓ Mostrar la cadena HOLA MAMA en letras gigantes

Luis Hernández Yáñez

Un dibujo

Luis Hernández Yáñez

Un dibujo

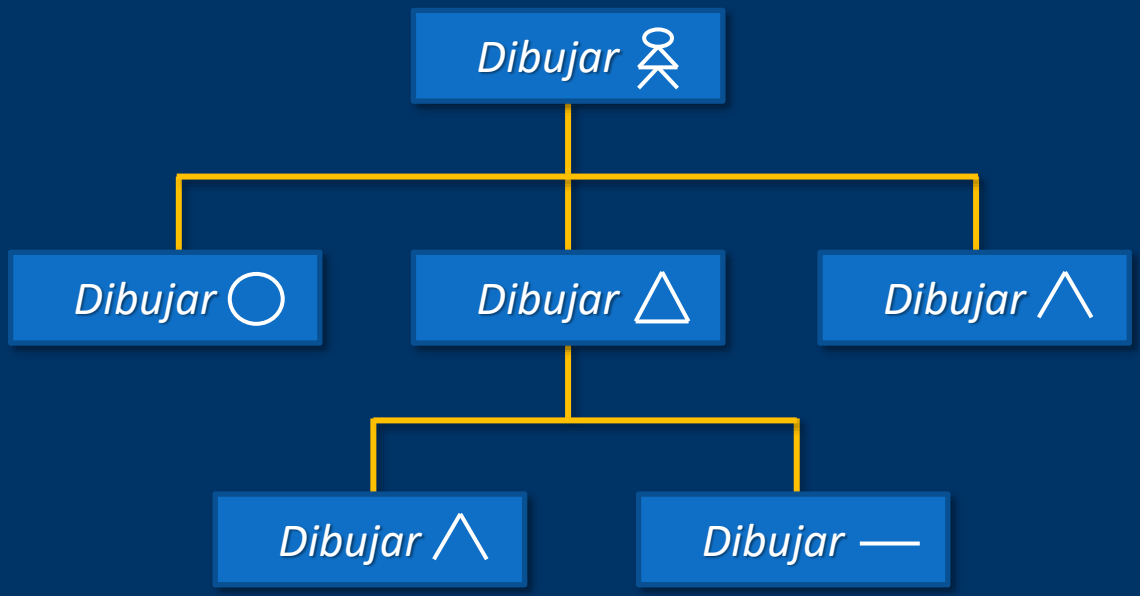
1. Dibujar 
2. Dibujar 
    2.1. Dibujar 
    2.2. Dibujar
3. Dibujar 

4 tareas, pero dos de ellas son iguales

Nos basta con saber cómo dibujar:

Luis Hernández Yáñez

Un dibujo

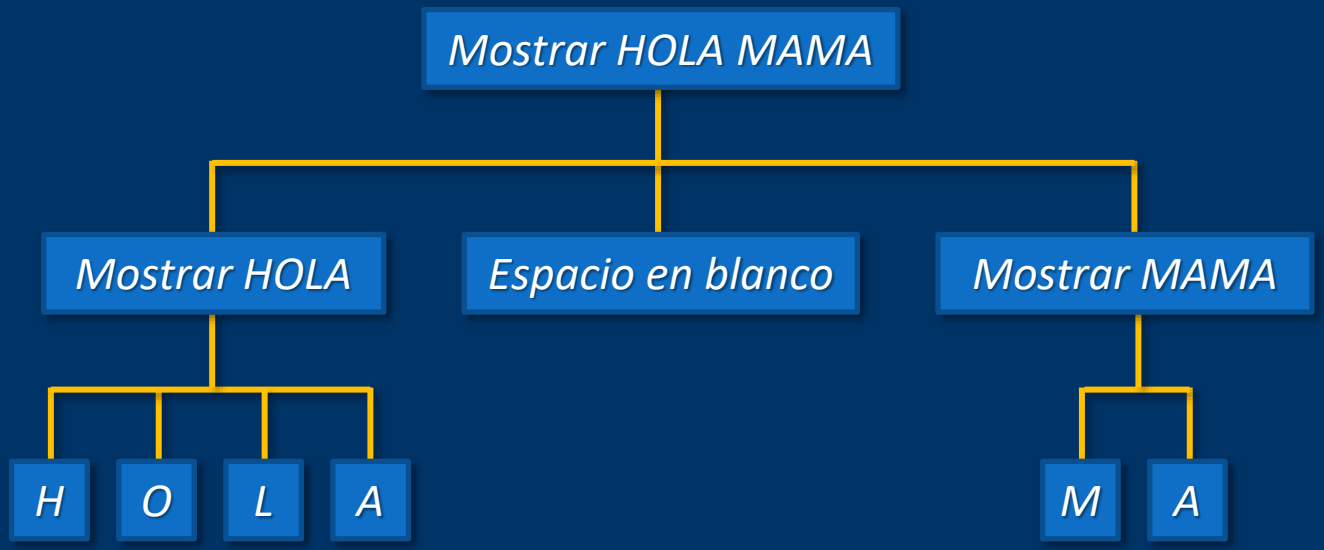



```
void dibujarCirculo()
{ ... }

void dibujarSecantes()
{ ... }

void dibujarLinea()
{ ... }

void dibujarTriangulo()
{
    dibujarSecantes();
    dibujarLinea();
}

int main() {
    dibujarCirculo();
    dibujarTriangulo();
    dibujarSecantes();
    return 0;
}
```

Luis Hernández Yáñez

Mensaje en letras gigantes

Mostrar la cadena HOLA MAMA en letras gigantes



Tareas básicas

Luis Hernández Yáñez

Mensaje en letras gigantes

```
void mostrarH() {
   cout << "*   *" << endl;
   cout << "*   *" << endl;
   cout << "*****" << endl;
   cout << "*   *" << endl;
   cout << "*   *" << endl << endl;
}

void mostrarO() {
   cout << "*****" << endl;
   cout << "*   *" << endl;
   cout << "*   *" << endl;
   cout << "*   *" << endl;
   cout << "*****" << endl << endl;
}

void mostrarL()
{ ... }

void mostrarA()
{ ...}
```

```
void espaciosEnBlanco() {
   cout << endl << endl << endl;
}

void mostrarM()
{ ...}

int main() {
   mostrarH();
   mostrarO();
   mostrarL();
   mostrarA();
   espaciosEnBlanco();
   mostrarM();
   mostrarA();
   mostrarM();
   mostrarA();

   return 0;
}
```

Luis Hernández Yáñez

Subprogramas

Luis Hernández Yáñez

Abstracción procedimental

Subprogramas

Pequeños programas dentro de otros programas

- ✓ Unidades de ejecución independientes
- ✓ Encapsulan código y datos
- ✓ Se comunican con otros subprogramas (datos)

Subrutinas, procedimientos, funciones, acciones, ...

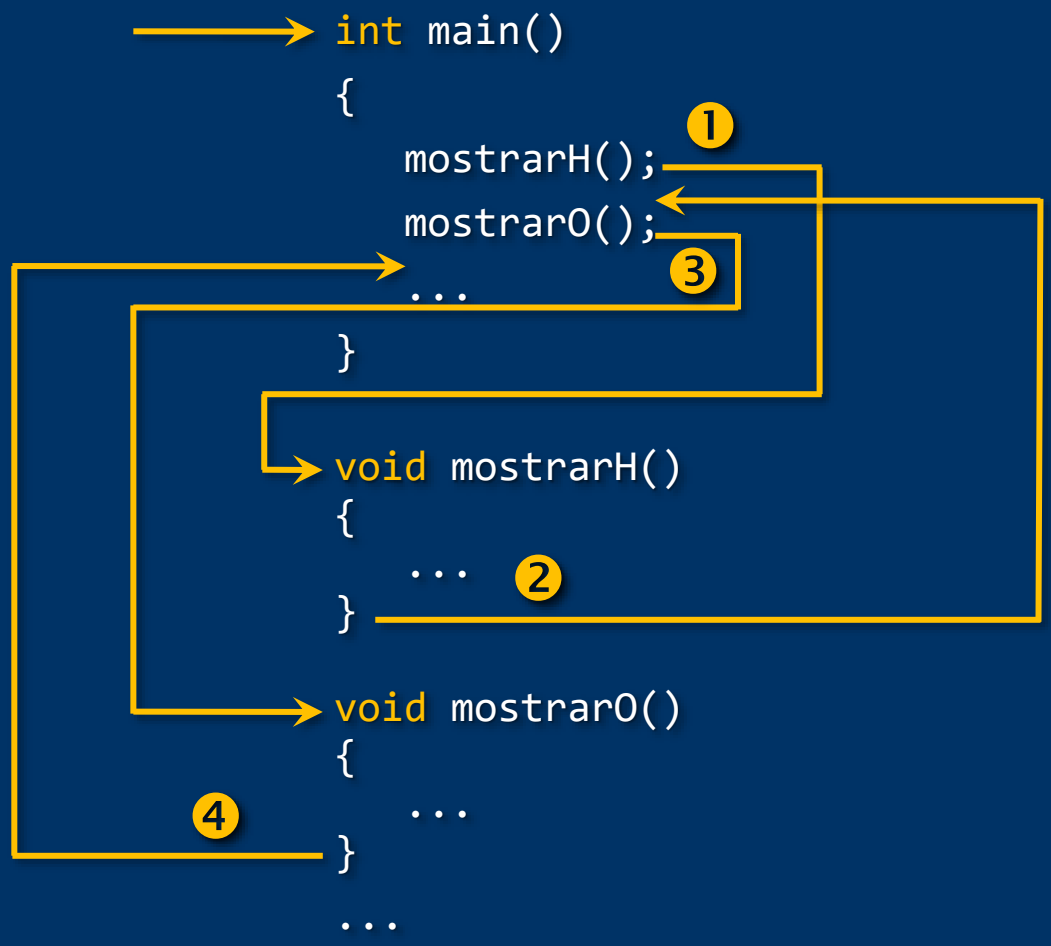
- ✓ Realizan tareas individuales del programa
- ✓ Funcionalidad concreta, identificable y coherente (diseño)
- ✓ Se ejecutan de principio a fin cuando se llaman (*invocan*)
- ✓ Terminan devolviendo el control al punto de llamada

Aumentan el nivel de abstracción del programa
Facilitan la prueba, la depuración y el mantenimiento

Luis Hernández Yáñez

Subprogramas

Flujo de ejecución



Luis Hernández Yáñez

Subprogramas

Subprogramas en C++

Forma general de un subprograma en C++:

```
tipo nombre(parámetros) // Cabecera
{
   // Cuerpo
}
```

- ✓ *Tipo* de dato que devuelve el subprograma como resultado
- ✓ *Parámetros* para la comunicación con el exterior
- ✓ *Cuerpo*: ¡Un bloque de código!

Luis Hernández Yáñez

Subprogramas

Tipos de subprogramas

Procedimientos (*acciones*):

- NO devuelven ningún resultado de su ejecución con `return`
- Tipo: `void`
- Llamada: instrucción independiente

```
mostrarH();
```

Funciones:

- SÍ devuelven un resultado con la instrucción `return`
- Tipo distinto de `void`
- Llamada: dentro de cualquier expresión

```
x = 12 * y + cuadrado(20) - 3;
```

- Se sustituye en la expresión por el valor que devuelve

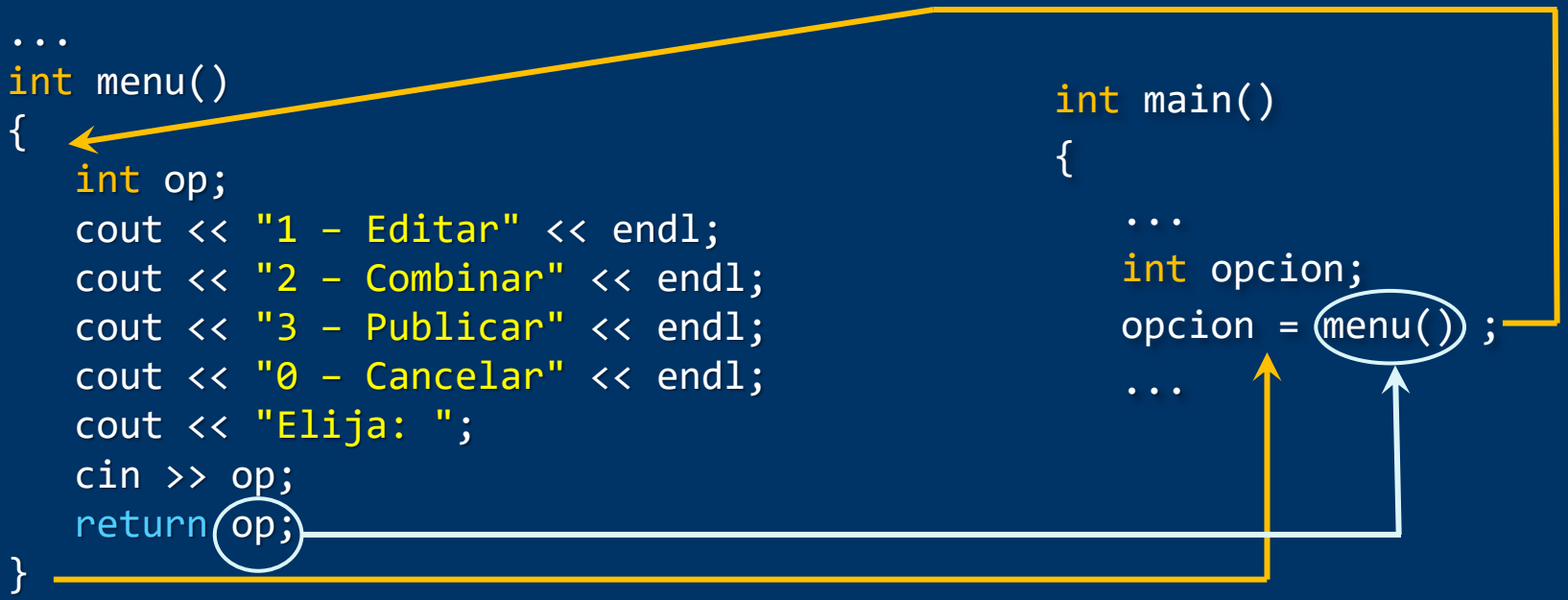
¡Ya venimos utilizando funciones desde el Tema 2!

Luis Hernández Yáñez

Subprogramas

Funciones

Subprogramas de tipo distinto de `void`



```
...
int menu()
{
   int op;
   cout << "1 - Editar" << endl;
   cout << "2 - Combinar" << endl;
   cout << "3 - Publicar" << endl;
   cout << "0 - Cancelar" << endl;
   cout << "Elija: ";
   cin >> op;
   return op;
}
```

```
int main()
{
   ...
   int opcion;
   opcion = menu() ;
   ...
```

Luis Hernández Yáñez

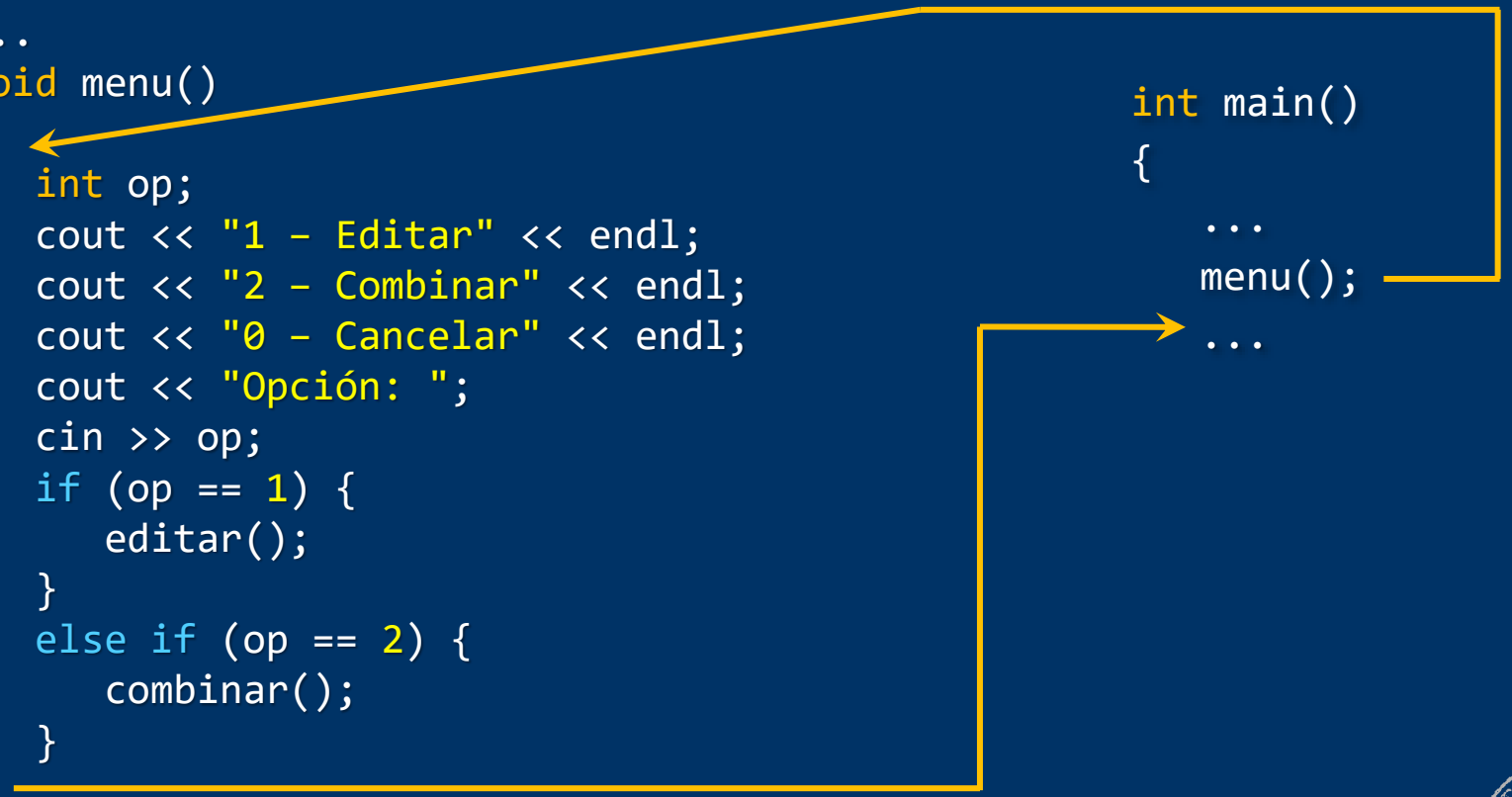
Subprogramas

Procedimientos

Subprogramas de tipo `void`

```
...
void menu()
{
   int op;
   cout << "1 - Editar" << endl;
   cout << "2 - Combinar" << endl;
   cout << "0 - Cancelar" << endl;
   cout << "Opción: ";
   cin >> op;
   if (op == 1) {
      editar();
   }
   else if (op == 2) {
      combinar();
   }
}
```

```
int main()
{
   ...
   menu();
   ...
```



Luis Hernández Yáñez

Subprogramas y datos

Luis Hernández Yáñez

Datos en los subprogramas

De uso exclusivo del subprograma

```
tipo nombre(parámetros) // Cabecera
{
   Declaraciones locales // Cuerpo
}
```

- ✓ Declaraciones locales de tipos, constantes y variables
  Dentro del cuerpo del subprograma
- ✓ Parámetros declarados en la cabecera del subprograma
  Comunicación del subprograma con otros subprogramas

Luis Hernández Yáñez

Datos locales y datos globales

Datos en los programas

- ✓ Datos globales: declarados fuera de todos los subprogramas
  Existen durante toda la ejecución del programa
- ✓ Datos locales: declarados en algún subprograma
  Existen sólo durante la ejecución del subprograma

Ámbito y visibilidad de los datos *Tema 3*

- Ámbito de los datos globales: resto del programa
  Se conocen dentro de los subprogramas que siguen
- Ámbito de los datos locales: resto del subprograma
  No se conocen fuera del subprograma
- Visibilidad de los datos
  Datos locales a un bloque ocultan otros externos homónimos

Luis Hernández Yáñez

Datos locales y datos globales

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

const int MAX = 100;
double ingresos;

...
void proc() {
   int op;
   double ingresos;
   ...
}

int main() {
   int op;
   ...
   return 0;
}
```

Datos globales

op de `proc()` es distinta de op de `main()`

Datos locales a `proc()`

Se conocen `MAX` (global), `op` (local) e `ingresos` (local que oculta la global)

Datos locales a `main()`

Se conocen `MAX` (global), `op` (local) e `ingresos` (global)

Luis Hernández Yáñez

Datos locales y datos globales

Sobre el uso de datos globales en los subprogramas

NO SE DEBEN USAR datos globales en subprogramas

- ✓ *¿Necesidad de datos externos?*

  Define parámetros en el subprograma

  Los datos externos se pasan como argumentos en la llamada

- ✓ Uso de datos globales en los subprogramas:

  Riesgo de *efectos laterales*

  Modificación inadvertida de esos datos afectando otros sitios

Excepciones:

- ✓ Constantes globales (valores inalterables)
- ✓ Tipos globales (necesarios en varios subprogramas)

Luis Hernández Yáñez

Parámetros

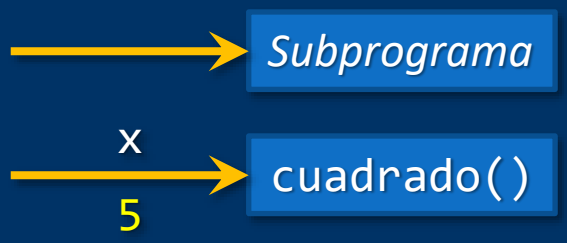
Luis Hernández Yáñez

Comunicación con el exterior

Datos de entrada, datos de salida y datos de entrada/salida

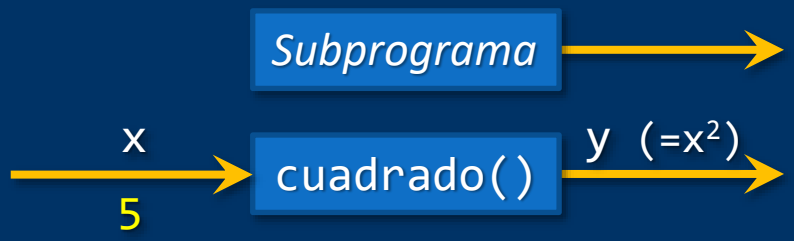
Datos de entrada: Aceptados

Subprograma que dado un número muestra en la pantalla su cuadrado:



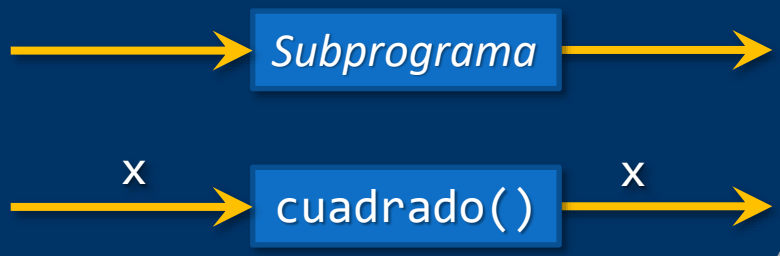
Datos de salida: Devueltos

Subprograma que dado un número devuelve su cuadrado:



Datos de entrada/salida: Aceptados y modificados

Subprograma que dada una variable numérica la eleva al cuadrado:



Luis Hernández Yáñez

Parámetros en C++

Declaración de parámetros

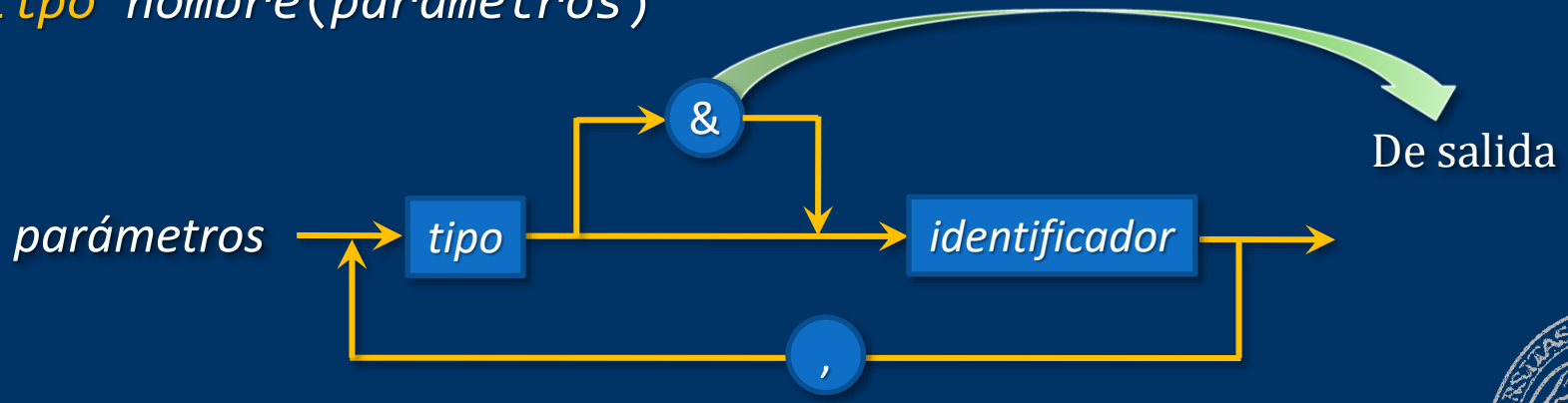
Sólo dos clases de parámetros en C++:

- Sólo de entrada (*por valor*)
- De salida (sólo salida o E/S) (*por referencia / por variable*)

Lista de parámetros formales

Entre los paréntesis de la cabecera del subprograma

```
tipo nombre(parámetros)
```



Luis Hernández Yáñez

Parámetros por valor

Reciben copias de los argumentos usados en la llamada

```
int cuadrado(int num)
```

```
double potencia(double base, int exp)
```

```
void muestra(string nombre, int edad, string nif)
```

```
void proc(char c, int x, double a, bool b)
```

Reciben sus valores en la llamada del subprograma

Argumentos: Expresiones en general

Variables, constantes, literales, llamadas a función, operaciones

Se destruyen al terminar la ejecución del subprograma

¡Atención! Los arrays se pasan por valor como constantes:

```
double media(const tArray lista)
```

Luis Hernández Yáñez

Parámetros por referencia &

Misma identidad que la variable pasada como argumento

```
void incrementa(int &x)
```

```
void intercambia(double &x, double &y)
```

```
void proc(char &c, int &x, double &a, bool &b)
```

Reciben las variables en la llamada del subprograma: *¡Variables!*

Los argumentos pueden quedar modificados

¡No usaremos parámetros por valor en las funciones!

Sólo en procedimientos

Puede haber tanto por valor como por referencia

¡Atención! Los arrays se pasan por referencia sin utilizar &

```
void insertar(tArray lista, int &contador, double item)
```

El argumento de `lista` (variable `tArray`) quedará modificado

Luis Hernández Yáñez

Argumentos

Luis Hernández Yáñez

Llamada a subprogramas con parámetros

nombre(*argumentos*)

- Tantos argumentos como parámetros y en el mismo orden
- Concordancia de tipos argumento-parámetro
- Por valor: Expresiones válidas (se pasa el resultado)
- Por referencia: *¡Sólo variables!*

Se copian los valores de las expresiones pasadas por valor en los correspondientes parámetros

Se hacen corresponder los argumentos pasados por referencia (variables) con sus correspondientes parámetros

Luis Hernández Yáñez

Argumentos pasados por valor

Expresiones válidas con concordancia de tipo:

```
void proc(int x, double a)
```

→
```
proc(23 * 4 / 7, 13.5);
```

→
```
double d = 3;
proc(12, d);
```

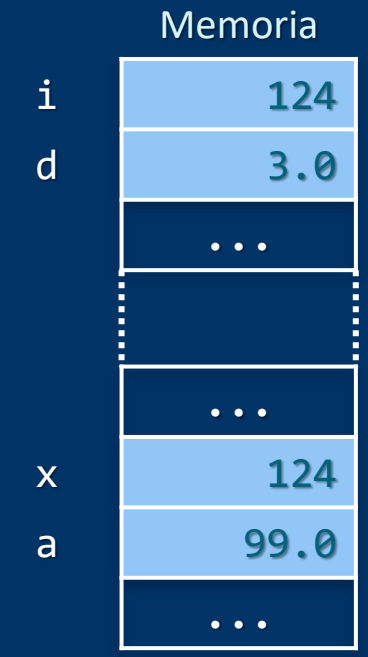
→
```
double d = 3;
int i = 124;
proc(i, 33 * d);
```

→
```
double d = 3;
int i = 124;
proc(cuad(20) * 34 + i, i * d);
```

Luis Hernández Yáñez

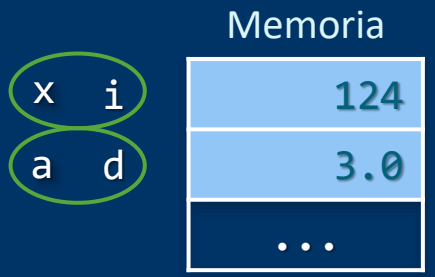
Argumentos pasados por valor

```
void proc(int x, double a)
{ ... }

int main()
{
  int i = 124;
  double d = 3;
  proc(i, 33 * d);
  ...

  return 0;
}
```



Argumentos pasados por referencia

```
void proc(int &x, double &a)
{ ... }

int main()
{
  int i = 124;
  double d = 3;
  proc(i, d);
  ...

  return 0;
}
```



¿Qué llamadas son correctas?

Dadas las siguientes declaraciones:

```
int i;
double d;
void proc(int x, double &a);
```

¿Qué pasos de argumentos son correctos? ¿Por qué no?

`proc(3, i, d);`	✗	Nº de argumentos ≠ Nº de parámetros
`proc(i, d);`	✓	
`proc(3 * i + 12, d);`	✓	
`proc(i, 23);`	✗	Parámetro por referencia → ¡variable!
`proc(d, i);`	✗	¡Argumento `double` para parámetro `int`!
`proc(3.5, d);`	✗	¡Argumento `double` para parámetro `int`!
`proc(i);`	✗	Nº de argumentos ≠ Nº de parámetros

Luis Hernández Yáñez

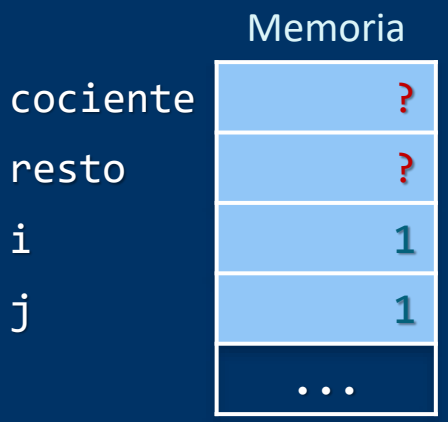
Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
   div = op1 / op2;
   rem = op1 % op2;
}

int main() {
   int cociente, resto;
   for (int j = 1; j <= 4; j++) {
      for (int i = 1; i <= 4; i++) {
         divide(i, j, cociente, resto);
         cout << i << " entre " << j << " da un cociente de "
            << cociente << " y un resto de " << resto << endl;
      }
   }

   return 0;
}
```

Luis Hernández Yáñez

Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
   div = op1 / op2;
   rem = op1 % op2;
}

int main() {
   int cociente, resto;
   for (int j = 1; j <= 4; j++) {
      for (int i = 1; i <= 4; i++) {
         divide(i, j, cociente, resto);
         ...
      }
   }

   return 0;
}
```



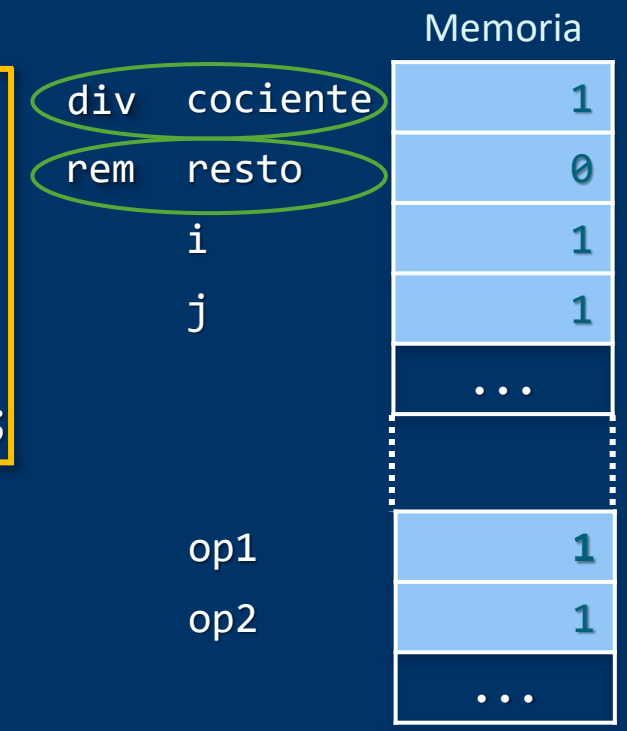
Luis Hernández Yáñez

Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
   div = op1 / op2;
   rem = op1 % op2;
}

int main() {
   int cociente, resto;
   for (int j = 1; j <= 4; j++) {
      for (int i = 1; i <= 4; i++) {
         divide(i, j, cociente, resto);
         ...
      }
   }

   return 0;
}
```

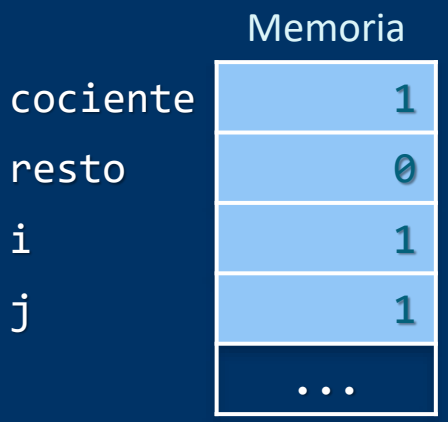
Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
   div = op1 / op2;
   rem = op1 % op2;
}

int main() {
   int cociente, resto;
   for (int j = 1; j <= 4; j++) {
      for (int i = 1; i <= 4; i++) {
         divide(i, j, cociente, resto);
         ...
      }
   }

   return 0;
}
```



Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
   div = op1 / op2;
   rem = op1 % op2;
}

int main() {
   int cociente, resto;
   for (int j = 1; j <= 4; j++) {
      for (int i = 1; i <= 4; i++) {
         divide(i, j, cociente, resto);
         ...
      }
   }

   return 0;
}
```

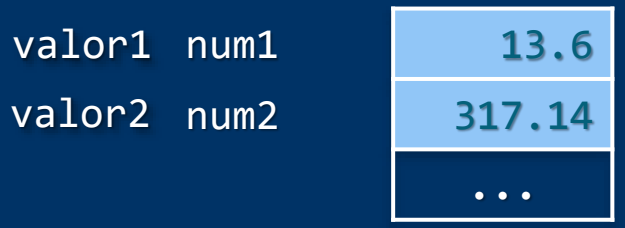


Más ejemplos

```
...
void intercambia(double &valor1, double &valor2) {
// Intercambia los valores
   double tmp; // Variable local (temporal)
   tmp = valor1;
   valor1 = valor2;
   valor2 = tmp;
}

int main() {
   double num1, num2;
   cout << "Valor 1: ";
   cin >> num1;
   cout << "Valor 2: ";
   cin >> num2;
   intercambia(num1, num2);
   cout << "Ahora el valor 1 es " << num1
        << " y el valor 2 es " << num2 << endl;
   return 0;
}
```

Memoria temporal del procedimiento

Memoria de main()



Más ejemplos

```
...
// Prototipo
void cambio(double precio, double pago, int &euros, int &cent50,
   int &cent20, int &cent10, int &cent5, int &cent2, int &cent1);

int main() {
   double precio, pago;
   int euros, cent50, cent20, cent10, cent5, cent2, cent1;
   cout << "Precio: ";
   cin >> precio;
   cout << "Pago: ";
   cin >> pago;
   cambio(precio, pago, euros, cent50, cent20, cent10, cent5, cent2,
          cent1);
   cout << "Cambio: " << euros << " euros, " << cent50 << " x 50c., "
        << cent20 << " x 20c., " << cent10 << " x 10c., "
        << cent5 << " x 5c., " << cent2 << " x 2c. y "
        << cent1 << " x 1c." << endl;

   return 0;
}
```

Luis Hernández Yáñez

Más ejemplos

```
void cambio(double precio, double pago, int &euros, int &cent50,
   int &cent20, int &cent10, int &cent5, int &cent2, int &cent1) {
   if (pago < precio) { // Cantidad insuficiente
      cout << "Error: El pago es inferior al precio" << endl;
   }
   else {
      int cantidad = int(100.0 * (pago - precio) + 0.5);
      euros  = cantidad / 100;
      cantidad = cambio % 100;
      cent50 = cantidad / 50;
      cantidad = cantidad % 50;
      cent20 = cantidad / 20;
      cantidad = cantidad % 20;
      cent10 = cantidad / 10;
      cantidad = cantidad % 10;
      cent5  = cantidad / 5;
      cantidad = cantidad % 5;
      cent2  = cantidad / 2;
      cent1  = cantidad % 2;
   }
}
```

Explicación en el libro de Adams/Leestma/Nyhoff

Luis Hernández Yáñez

Notificación de errores

En los subprogramas se pueden detectar errores

Errores que impiden realizar los cálculos:

```
void cambio(double precio, double pago, int &euros, int &cent50,
   int &cent20, int &cent10, int &cent5, int &cent2, int &cent1) {
   if (pago < precio) { // Cantidad insuficiente
      cout << "Error: El pago es inferior al precio" << endl;
   }
   ...
```

¿Debe el subprograma notificar al usuario o al programa?

→ Mejor notificarlo al punto de llamada y allí decidir qué hacer

```
void cambio(double precio, double pago, int &euros, int &cent50,
   int &cent20, int &cent10, int &cent5, int &cent2, int &cent1,
   bool &error) {
   if (pago < precio) { // Cantidad insuficiente
      error = true;
   }
   else {
      error = false;
      ...
```

Luis Hernández Yáñez

Notificación de errores

cambio.cpp

Al volver de la llamada se decide qué hacer si ha habido error...

- ✓ ¿Informar al usuario?
- ✓ ¿Volver a pedir los datos?
- ✓ Etcétera

```
int main() {
   double precio, pago;
   int euros, cent50, cent20, cent10, cent5, cent2, cent1;
   bool error;
   cout << "Precio: ";
   cin >> precio;
   cout << "Pago: ";
   cin >> pago;
   cambio(precio, pago, euros, cent50, cent20, cent10, cent5, cent2,
          cent1, error);
   if (error) {
      cout << "Error: El pago es inferior al precio" << endl;
   }
   else {
      ...
```

Luis Hernández Yáñez

Resultado de la función

Luis Hernández Yáñez

Resultado de la función

Una función ha de devolver un resultado

La función ha de terminar su ejecución devolviendo el resultado

La instrucción `return`:

- Devuelve el dato que se indica a continuación como resultado
- Termina la ejecución de la función

El dato devuelto sustituye a la llamada de la función en la expresión

```
int cuad(int x) {
   return x * x;
   x = x * x;
}
```

Esta instrucción no se ejecutará nunca

```
int main() {
   cout << 2 * cuad(16);
   return 0;
}
```

256

Luis Hernández Yáñez

Ejemplo: Cálculo del factorial

factorial.cpp

Factorial (N) = 1 x 2 x 3 x ... x (N-2) x (N-1) x N

```
long long int factorial(int n); // Prototipo

int main() {
   int num;
   cout << "Num: ";
   cin >> num;
   cout << "Factorial de " << num << ": " << factorial(num) << endl;
   return 0;
}

long long int factorial(int n) {
   long long int fact = 1;
   if (n < 0) {
      fact = 0;
   }
   else {
      for (int i = 1; i <= n; i++) {
         fact = fact * i;
      }
   }
   return fact;
}
```

Luis Hernández Yáñez

Un único punto de salida

```
int compara(int val1, int val2) {
// -1 si val1 < val2, 0 si iguales, +1 si val1 > val2
   if (val1 == val2) {
      return 0;
   }
   else if (val1 < val2) {
      return -1;
   }
   else {
      return 1;
   }
}
```

¡3 puntos de salida!

Luis Hernández Yáñez

Un único punto de salida

```
int compara(int val1, int val2) {
// -1 si val1 < val2, 0 si iguales, +1 si val1 > val2
   int resultado;

   if (val1 == val2) {
      resultado = 0;
   }
   else if (val1 < val2) {
      resultado = -1;
   }
   else {
      resultado = 1;
   }

   return resultado;
}
```

Punto de salida único

Luis Hernández Yáñez

¿Cuándo termina el subprograma?

Procedimientos (tipo `void`):

- Al encontrar la llave de cierre que termina el subprograma o
- Al encontrar una instrucción `return` (sin resultado)

Funciones (tipo distinto de `void`):

- SÓLO al encontrar una instrucción `return` (con resultado)

Nuestros subprogramas siempre terminarán al final:

- ✓ No usaremos `return` en los procedimientos
- ✓ Funciones: sólo un `return` y estará al final

Para facilitar la depuración y el mantenimiento, codifica los subprogramas con un único punto de salida

Luis Hernández Yáñez

Prototipos

Luis Hernández Yáñez

¿Qué subprogramas hay en el programa?

¿Dónde los ponemos? ¿Antes de `main()`? ¿Después de `main()`?

→ Los pondremos después de `main()`

¿Son correctas las llamadas a subprogramas?

En `main()` o en otros subprogramas

- ¿Existe el subprograma?
- ¿Concuerdan los argumentos con los parámetros?

Deben estar los prototipos de los subprogramas antes de `main()`

Prototipo: cabecera del subprograma terminada en `;`

```
void dibujarCirculo();
void mostrarM();
void proc(double &a);
int cuad(int x);
...
```

`main()` es el único subprograma que no hay que prototipar

Luis Hernández Yáñez

Ejemplos

intercambia.cpp

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

void intercambia(double &valor1, double &valor2); // Prototipo

int main() {
   double num1, num2;
   cout << "Valor 1: ";
   cin >> num1;
   cout << "Valor 2: ";
   cin >> num2;
   intercambia(num1, num2);
   cout << "Ahora el valor 1 es " << num1
        << " y el valor 2 es " << num2 << endl;
   return 0;
}

void intercambia(double &valor1, double &valor2) {
   double tmp; // Variable local (temporal)
   tmp = valor1;
   valor1 = valor2;
   valor2 = tmp;
}
```

Asegúrate de que los prototipos coincidan con las implementaciones

Luis Hernández Yáñez

Ejemplos

mates.cpp

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

// Prototipos
long long int factorial(int n);
int sumatorio(int n);

int main() {
   int num;
   cout << "Num: ";
   cin >> num;
   cout << "Factorial de "
        << num << ": "
        << factorial(num)
        << endl
        << "Sumatorio de 1 a "
        << num << ": "
        << sumatorio(num)
        << endl;

   return 0;
}
```

```cpp
long long int factorial(int n) {
   long long int fact = 1;

   if (n < 0) {
      fact = 0;
   }
   else {
      for (int i = 1; i <= n; i++) {
         fact = fact * i;
      }
   }

   return fact;
}

int sumatorio(int n) {
   int sum = 0;

   for (int i = 1; i <= n; i++) {
      sum = sum + i;
   }

   return sum;
}
```

Luis Hernández Yáñez

Funciones de operador

Luis Hernández Yáñez

Funciones de operador

Notación infija (de operador)

operandoIzquierdo *operador* *operandoDerecho*

```
a + b
```

Se ejecuta el operador con los operandos como argumentos

Los operadores se implementan como funciones:

```
tipo operatorsímbolo(parámetros)
```

Si es un operador monario sólo habrá un parámetro

Si es binario habrá dos parámetros

El *símbolo* es un símbolo de operador (uno o dos caracteres):

+, -, *, /, --, <<, %, ...

Luis Hernández Yáñez

Funciones de operador

```
tMatriz suma(tMatriz a, tMatriz b);
```

```
tMatriz a, b, c;
c = suma(a, b);
```

```
tMatriz operator+(tMatriz a, tMatriz b);
```

```
tMatriz a, b, c;
c = a + b;
```

¡La implementación será exactamente la misma!

Mayor aproximación al lenguaje matemático

Luis Hernández Yáñez

Diseño descendente (un ejemplo)

Luis Hernández Yáñez

Refinamientos sucesivos

Especificación inicial (Paso 0).-

Desarrollar un programa que haga operaciones de conversión de medidas hasta que el usuario decida que no quiere hacer más

Análisis y diseño aumentando el nivel de detalle en cada paso

¿Qué operaciones de conversión?

Paso 1.-

Desarrollar un programa que haga operaciones de conversión de medidas hasta que el usuario decida que no quiere hacer más

* *Pulgadas a centímetros*
* *Libras a gramos*
* *Grados Fahrenheit a centígrados*
* *Galones a litros*

Luis Hernández Yáñez

Refinamientos sucesivos

Paso 2.-

Desarrollar un programa que muestre al usuario un menú con cuatro operaciones de conversión de medidas:

* *Pulgadas a centímetros*
* *Libras a gramos*
* *Grados Fahrenheit a centígrados*
* *Galones a litros*

Y lea la elección del usuario y proceda con la conversión, hasta que el usuario decida que no quiere hacer más

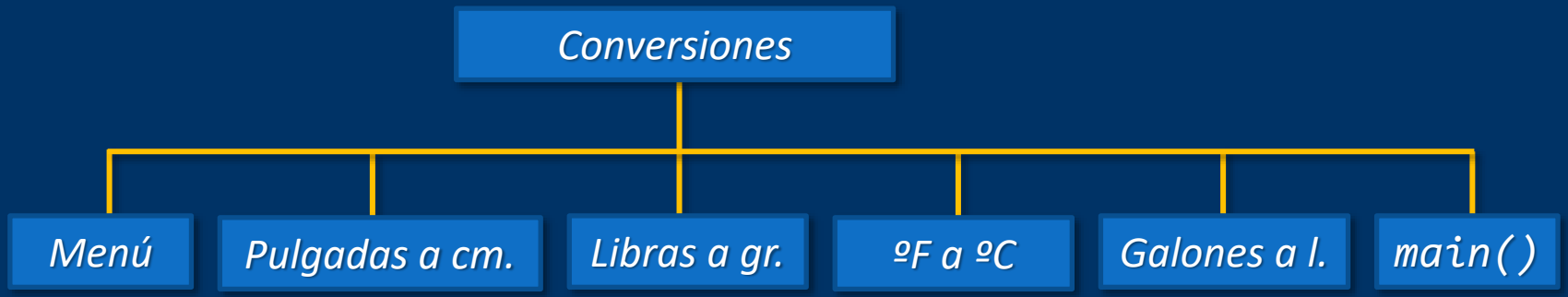
6 grandes tareas:

Menú, cuatro funciones de conversión y `main()`

Luis Hernández Yáñez

Refinamientos sucesivos

Paso 2.-



Luis Hernández Yáñez

Refinamientos sucesivos

Paso 3.-

* *Menú:*
  Mostrar las cuatro opciones más una para salir
  Validar la entrada y devolver la elegida
* *Pulgadas a centímetros:*
  Devolver el equivalente en centímetros del valor en pulgadas
* *Libras a gramos:*
  Devolver el equivalente en gramos del valor en libras
* *Grados Fahrenheit a centígrados:*
  Devolver el equivalente en centígrados del valor en Fahrenheit
* *Galones a litros:*
  Devolver el equivalente en litros del valor en galones
* *Programa principal* (`main()`)

Luis Hernández Yáñez

Refinamientos sucesivos

Paso 3.- Cada tarea, un subprograma

Comunicación entre los subprogramas:

Función	Entrada	Salida	Valor devuelto
`menu()`	–	–	`int`
`pulgACm()`	`double pulg`	–	`double`
`lbAGr()`	`double libras`	–	`double`
`grFAGrC()`	`double grF`	–	`double`
`galALtr()`	`double galones`	–	`double`
`main()`	–	–	`int`

Luis Hernández Yáñez

Refinamientos sucesivos

Paso 4.- Algoritmos detallados de cada subprograma → Programar

```
#include <iostream>
using namespace std;
// Prototipos
int menu();
double pulgACm(double pulg);
double lbAGr(double libras);
double grFAGrC(double grF);
double galALtr(double galones);

int main() {
   double valor;
   int op = -1;
   while (op != 0) {
      op = menu();
      switch (op) {
      case 1:
         {
            cout << "Pulgadas: ";
            cin >> valor;
            cout << "Son " << pulgACm(valor) << " cm." << endl;
         }
         break;
                                                              ...
```

Luis Hernández Yáñez

Refinamientos sucesivos

```
        case 2:
            {
                cout << "Libras: ";
                cin >> valor;
                cout << "Son " << lbAGr(valor) << " gr." << endl;
            }
            break;
        case 3:
            {
                cout << "Grados Fahrenheit: ";
                cin >> valor;
                cout << "Son " << grFAGrC(valor) << " ºC" << endl;
            }
            break;
        case 4:
            {
                cout << "Galones: ";
                cin >> valor;
                cout << "Son " << galALtr(valor) << " l." << endl;
            }
            break;
        }
    }
    return 0;
}
```

...

Luis Hernández Yáñez

Refinamientos sucesivos

```
int menu() {
   int op = -1;

   while ((op < 0) || (op > 4)) {
      cout << "1 - Pulgadas a Cm." << endl;
      cout << "2 - Libras a Gr." << endl;
      cout << "3 - Fahrenheit a ºC" << endl;
      cout << "4 - Galones a L." << endl;
      cout << "0 - Salir" << endl;
      cout << "Elige: ";
      cin >> op;
      if ((op < 0) || (op > 4)) {
         cout << "Opción no válida" << endl;
      }
   }

   return op;
}

double pulgACm(double pulg) {
   const double cmPorPulg = 2.54;
   return pulg * cmPorPulg;
}
```

...

Luis Hernández Yáñez

Refinamientos sucesivos

conversiones.cpp

```
double lbAGr(double libras) {
   const double grPorLb = 453.6;
   return libras * grPorLb;
}

double grFAGrC(double grF) {
   return ((grF - 32) * 5 / 9);
}

double galALtr(double galones) {
   const double ltrPorGal = 4.54609;
   return galones * ltrPorGal;
}
```

...

Luis Hernández Yáñez

Precondiciones y postcondiciones

Luis Hernández Yáñez

Precondiciones y postcondiciones

Integridad de los subprogramas

Condiciones que se deben dar antes de comenzar su ejecución

→ Precondiciones

✓ Quien llame al subprograma debe garantizar que se satisfacen

Condiciones que se darán cuando termine su ejecución

→ Postcondiciones

✓ En el punto de llamada se pueden dar por garantizadas

Aserciones:

Condiciones que si no se cumplen interrumpen la ejecución

Función `assert()`

Luis Hernández Yáñez

Aserciones como precondiciones

Precondiciones

Por ejemplo, no realizaremos conversiones de valores negativos:

```
double pulgACm(double pulg) {
   assert(pulg > 0);

   double cmPorPulg = 2.54;

   return pulg * cmPorPulg;
}
```

La función tiene una precondición: `pulg` debe ser positivo

`assert(pulg > 0);` interrumpirá la ejecución si no es cierto

Luis Hernández Yáñez

Aserciones como precondiciones

Precondiciones

Es responsabilidad del punto de llamada garantizar la precondición:

```
int main() {
   double valor;
   int op = -1;
   while (op != 0) {
      op = menu();
      switch (op) {
      case 1:
         {
            cout << "Pulgadas: ";
            cin >> valor;
            if (valor < 0) {
               cout << "¡No válido!" << endl;
            }
            else { // Se cumple la precondición...
               ...
```

Luis Hernández Yáñez

Aserciones como postcondiciones

Postcondiciones

Un subprograma puede garantizar condiciones al terminar:

```
int menu() {
   int op = -1;
   while ((op < 0) || (op > 4)) {
      ...
      cout << "Elige: ";
      cin >> op;
      if ((op < 0) || (op > 4)) {
         cout << "Opción no válida" << endl;
      }
   }
   assert ((op >= 0) && (op <= 4));
   return op;
}
```

El subprograma debe asegurarse de que se cumpla

Luis Hernández Yáñez

Acerca de *Creative Commons*

Licencia CC (Creative Commons)

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

- Reconocimiento (*Attribution*):
  En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
- No comercial (*Non commercial*):
  La explotación de la obra queda limitada a usos no comerciales.
- Compartir igual (*Share alike*):
  La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.