Fundamentos de programación

4 La abstracción procedimental

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

Material de la Prof.ª Mercedes Gómez Albarrán
Versión revisada y ampliada del material del Prof. Luis Hernández Yáñez

Facultad de Informática
Universidad Complutense

Diseño descendente y abstracción procedimental

Diseño descendente / Refinamientos sucesivos

Un programa debe realizar una serie de tareas.

Cada tarea se podrá subdividir en subtareas más sencillas.

Las subtareas también se podrán subdividir en otras más sencillas ...

Mercedes Gómez Albarrán

Diseño descendente y abstracción procedimental

Dibujar figura

1. Dibujar 
2. Dibujar 
3. Dibujar 

REFINAMIENTO

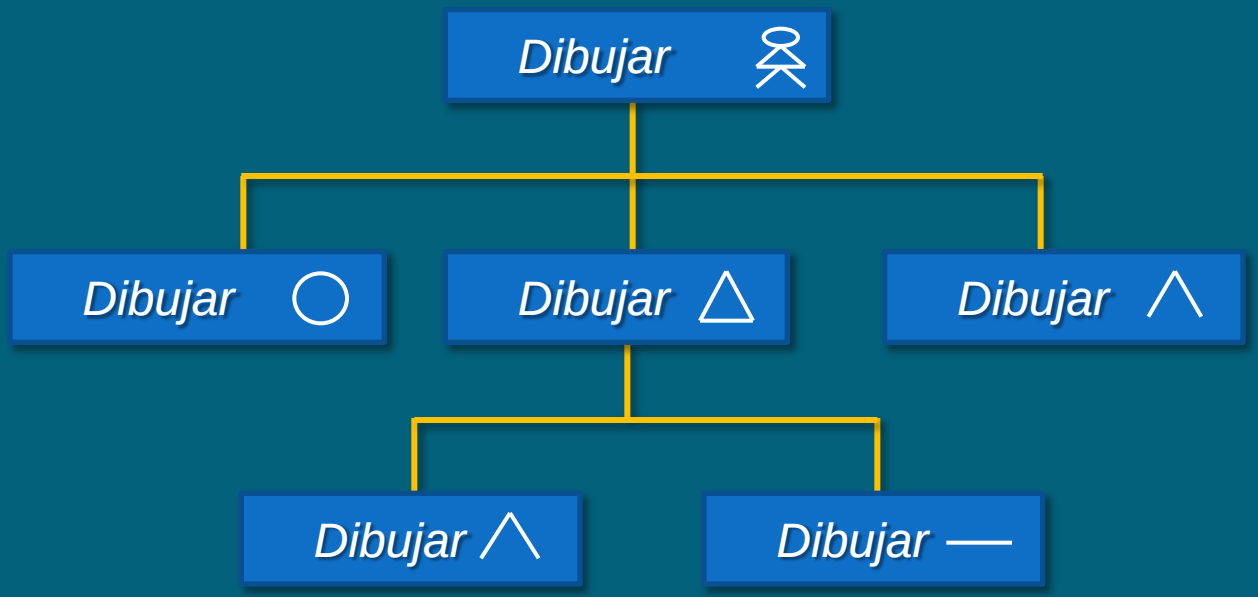
1. Dibujar 
2. Dibujar 
   - 2.1. Dibujar 
   - 2.2. Dibujar
3. Dibujar 

Misma tarea

Mercedes Gómez Albarrán

Diseño descendente y abstracción procedimental

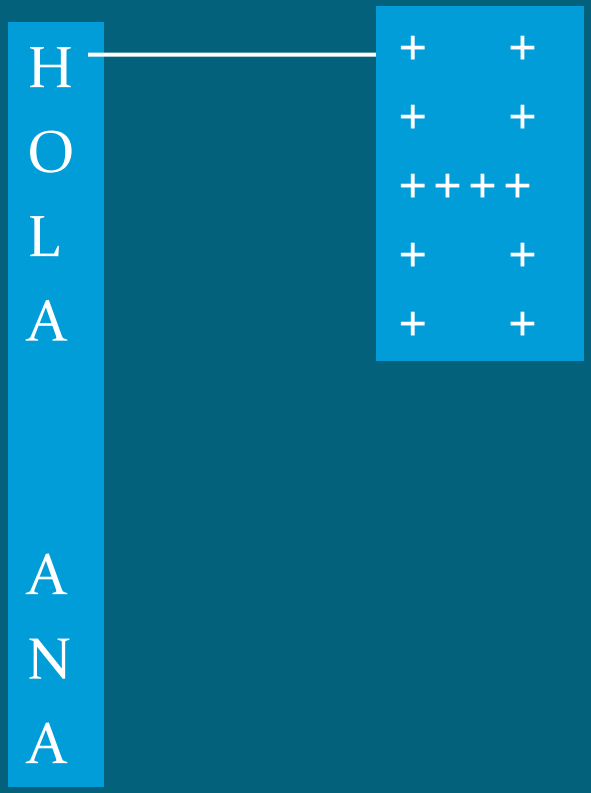
Dibujar figura



Mercedes Gómez Albarrán

Diseño descendente y abstracción procedimental

Mensaje en letras gigantes



- Imprimir mensaje
  - Imprimir HOLA
    - Imprimir H
    - Imprimir O
    - Imprimir L
    - Imprimir A
  - Imprimir dos líneas en blanco
  - Imprimir ANA
    - Imprimir A
    - Imprimir N
    - Imprimir A

Mercedes Gómez Albarrán

Diseño descendente y abstracción procedimental

Mensaje en letras gigantes

Mercedes Gómez Albarrán

Abstracción procedimental: subprogramas

Subprograma = módulo de código y datos que lleva a cabo una funcionalidad concreta (objetivo del subprograma)

- Precondiciones: condiciones que se deben dar antes de comenzar su ejecución
- Poscondiciones: condiciones que darán al finalizar su ejecución
- Puede comunicarse con *el exterior*
- Al terminar su ejecución devuelve el control al punto donde se efectuó la llamada

Los subprogramas:

-Aumentan el nivel de abstracción del programa.

-Promueven la reutilización.

-Facilitan la prueba, la depuración y el mantenimiento.

Mercedes Gómez Albarrán

Abstracción procedimental: subprogramas

Integridad de los subprogramas

Un subprograma puede establecer una serie de → Precondiciones

- Será responsabilidad del que llame al subprograma garantizar que se satisfacen las precondiciones

El subprograma puede, además, garantizar otra serie de condiciones que se darán cuando termine su ejecución → Postcondiciones

- En el punto de llamada se pueden dar por garantizadas las postcondiciones

Mercedes Gómez Albarrán

Abstracción procedimental: subprogramas

Datos manejados por un subprograma

El ámbito de los identificadores de los datos locales y de los parámetros es el código del subprograma → de uso exclusivo del subprograma

- Datos locales (variables y constantes)
  Declarados dentro del cuerpo del subprograma
- Datos intercambiados con *el exterior* → El concepto de **parámetro**

Datos de entrada: aceptados

Datos de salida: devueltos

Datos de entrada/salida:
aceptados y modificados

Mercedes Gómez Albarrán

Abstracción procedimental: subprogramas

Datos manejados por un subprograma: Comunicación con el exterior

Subprograma que dado un número calcula y muestra en pantalla su cuadrado:

Subprograma que dado un número calcula y devuelve su cuadrado:

Subprograma que dada una variable numérica la devuelva incrementada en 1:

Mercedes Gómez Albarrán

Subprogramas en C++

Forma general

```
Tipo Nombre(Parámetros) // Cabecera
{
  // Cuerpo
}
```

- *Tipo*: tipo del dato en el que se transforma la llamada al subprograma como resultado de su ejecución (tipo del resultado)
- *Parámetros* para intercambiar información con el exterior
- *Cuerpo*: secuencia de declaraciones locales e instrucciones.

Mercedes Gómez Albarrán

Subprogramas en C++

Tipos de subprogramas

- Procedimientos: subprogramas con Tipo `void`

  No devuelven ningún resultado de su ejecución.
  Llamada: instrucción independiente.

- Funciones: subprogramas con Tipo distinto de `void`

  Devuelven un resultado.
  Llamada: en cualquier punto donde pueda ir un valor del tipo del subprograma.

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

Mercedes Gómez Albarrán

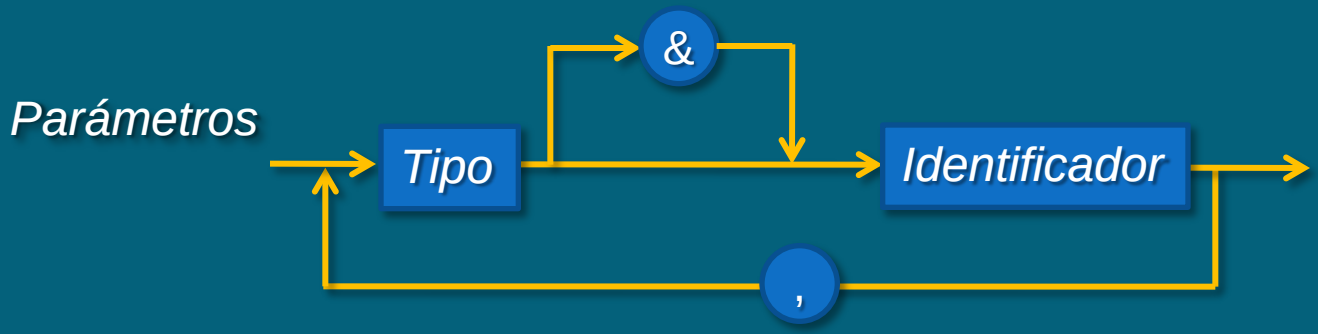
Subprogramas en C++

Parámetros

Se distinguen dos tipos de parámetros:

- Parámetros por valor → datos de entrada
- Parámetros por referencia o por variable → datos de salida o E/S

Se declaran en la cabecera, a continuación del nombre de la función y entre paréntesis `Tipo Nombre(Parámetros)`



Mercedes Gómez Albarrán

Subprogramas en C++

Ejemplos de parámetros por valor y por referencia

```
void func1(int x)

void func2(int x, int y, int z, double& a)

bool func3(int x, int y, int& z, int& w)

int func4(float& a, int x)
```

Mercedes Gómez Albarrán

Subprogramas en C++

Llamada (o invocación) a subprograma

```
Nombre(Argumentos)
```

- Deben ponerse entre los paréntesis tantos argumentos como parámetros
- Debe respetarse el orden de declaración de los parámetros
- Cada argumento, del mismo tipo que el correspondiente parámetro
- Argumento para un parámetro por valor: una constante, una variable o una expresión
- Argumento para un parámetro por referencia/variable: sólo puede ser una variable

Mercedes Gómez Albarrán

Subprogramas en C++

Ejemplo:

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


func(3, i, d);
func(i, d);
func(3 * i + 12, d);
func(i, 23);
func(d, i);
func(3.5, d);
func(i);
```

¿Qué llamadas son correctas?

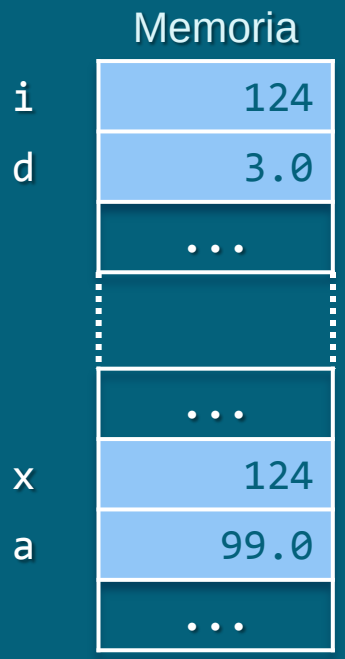
Mercedes Gómez Albarrán

Paso de argumentos con parámetros por valor

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


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

  return 0;
}
```



Paso de argumentos con parámetros por referencia

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


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

  return 0;
}
```

Mercedes Gómez Albarrán

Procedimientos en C++

- Subprogramas con Tipo `void`
- La llamada no se transforma en ningún valor (tipo de devolución `void`)
- Llamada: instrucción independiente

```
void Nombre(Parámetros) // Cabecera
  {
    // Cuerpo
  }
```

Mercedes Gómez Albarrán

Procedimientos en C++

Ejemplo 1.- Procedimiento que calcula el área de un triángulo dadas su base y su altura


```
void calculaArea (float b, float a, float& areaTriangulo)
{
     areaTriangulo = b * a / 2;
}
```

Mercedes Gómez Albarrán

Procedimientos en C++

Ejemplo 1.- Procedimiento que calcula el área de un triángulo dadas su base y su altura

```
int main()
{
  float base, altura, area;
  cout << "Introduzca la base y la altura del triángulo: ";
  cin >> base;
  cin >> altura;
  if ( altura > 0 && base > 0 ) {
     calculaArea(base, altura, area);
     cout << "El área de un triángulo es: " << area << endl;
  }
  else   cout << "Datos erroneos!";
  return 0;
}
```

¿Qué ocurre en memoria?

Procedimientos en C++

Ejemplo 1.- Procedimiento que calcula el área de un triángulo dadas su base y su altura

```
int main()
{
  float base, altura, area;
  cout << "Introduzca la base y la altura del triángulo: ";
  cin >> base;
  cin >> altura;
  if ( altura > 0 && base > 0 ) {
    cout << "El área de un triángulo es: ";
    cout <<  calculaArea(base, altura, area) << endl;
  }
  else   cout << "Datos erroneos!";
  return 0;
}
```

¿Qué está mal en este código?

Procedimientos en C++

Ejemplo 2.- Procedimiento que intercambia dos datos reales dados


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

intercambio.cpp

Mercedes Gómez Albarrán

Procedimientos en C++

Ejemplo 2.- Procedimiento que intercambia dos datos reales dados

```
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;
}
```

¿Qué ocurre en memoria?

Mercedes Gómez Albarrán

Procedimientos en C++

Ejemplo 2.- Procedimiento que intercambia dos datos reales dados

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

int main() {
  double valor1, valor2;
  cout << "Valor 1: "; cin >> valor1;
  cout << "Valor 2: "; cin >> valor2;
  intercambia(valor1, valor2);
  cout << "Ahora el valor 1 es " << valor1
       << " y el valor 2 es " << valor2 << endl;
  return 0;
}
```

¿Se pueden llamar igual los parámetros y los argumentos?

Mercedes Gómez Albarrán

Procedimientos en C++

Ejemplo 3.- Procedimiento que calcula la suma de los enteros entre 1 y un número entero positivo dado

$$\sum_{i=1}^{N} i$$

Mercedes Gómez Albarrán

Procedimientos en C++

Ejemplo 4.- Procedimiento que calcula la suma de los números reales que hay en el archivo dado

```
void sumatorio_archivo (ifstream& arch, float& resul)  {
   float dato;

   resul = 0;
   arch >> dato;
   while (dato != 0) { // se supone que 0 es el centinela
     resul = resul + dato;
     arch >> dato;
   }
}
```

Mercedes Gómez Albarrán

Procedimientos en C++

Ejemplo 4.- Procedimiento que calcula la suma de los números reales que hay en el archivo dado

```
int main()  {
  float resultado;
  ifstream archivo;

  archivo.open("datos.txt");
  if (!archivo.is_open())
    cout << "ERROR DE APERTURA" << endl;
  else {
     sumatorio_archivo (archivo, resultado);
     cout << "Suma = " << resultado << endl;
     archivo.close();
  }
return 0;
}
```

Mercedes Gómez Albarrán

Funciones en C++

- Subprogramas con Tipo distinto de `void`
- La llamada se transforma en un valor del tipo de devolución

  La instrucción `return`

  — Devuelve el dato que se indique a continuación como resultado

  — Termina la ejecución de la función

  En una función puede haber más de una instrucción `return`
- Llamada: en cualquier punto donde pueda ir un valor del tipo del subprograma
- Las funciones suelen reservarse para cuando el subprograma devuelve un dato y sólo uno

Mercedes Gómez Albarrán

Ejemplo 5.- Función que calcula el área de un triángulo dadas su base y su altura


```
float calculaArea (float b, float a)  {
     return b * a / 2;
}

float calculaArea (float b, float a)  {
     float area;
     area = b * a / 2;
     return area;
}
```

¿Válidas ambas versiones?

Mercedes Gómez Albarrán

Ejemplo 5.- Función que calcula el área de un triángulo dadas su base y su altura

```
int main()
{
  float base, altura;
  cout << "Introduzca la base y la altura del triangulo: ";
  cin >> base;  cin >> altura;
  if ( altura > 0 && base > 0 )
    cout << "El area de un triangulo es: "
         << calculaArea(base, altura) << endl;
  else
    cout << "Datos erroneos!";
  return 0;
}
```

Mercedes Gómez Albarrán

Ejemplo 5.- Función que calcula el área de un triángulo dadas su base y su altura

triangulo.cpp

```
int main()
{
  float base, altura, resultado;
  cout << "Introduzca la base y la altura del triangulo: ";
  cin >> base;   cin >> altura;
  if ( altura > 0 && base > 0 )  {
    resultado =  calculaArea(base, altura);
    cout << "El area de un triangulo es: " << resultado << endl;
  }
  else
    cout << "Datos erroneos!";
  return 0;
}
```

¿Es correcta esta segunda versión del programa?

Ejemplo 6.- Función que calcula la suma de los enteros entre 1 y un número entero positivo dado

$$\sum_{i=1}^{N} i$$

sumatorio.cpp

```
int sumatorio (int N)  {
   int resultado = 0;
   for(int i = 1; i <= N; i++)
    resultado = resultado + i;
   return resultado;
}
```

Mercedes Gómez Albarrán

Ejemplo 7.- Función que indica si 2 números dados son iguales, si el primero es mayor que el segundo o viceversa


```
int comparacion(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!

```
int comparacion(int val1, int val2)
{
  int resultado;
  if (val1 == val2) resultado = 0;
  else if (val1 < val2) resultado = -1;
  else resultado = 1;
  return resultado;
}
```

Punto de salida único

Funciones en C++

Ejemplo 7.- Función que indica si 2 números dados son iguales, si el primero es mayor que el segundo o viceversa

```
int comparacion(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;
  cout << "Fin de la función!!";

}
```

¿Qué ocurre en cualquier ejecución de esta función?

Mercedes Gómez Albarrán

Funciones en C++: la función `main`

Comunicación con el sistema operativo: Parámetros de `main`

Permiten obtener datos proporcionados al ejecutar el programa desde la línea de órdenes

```
int main(int argc, char* argv[])
```

Parámetros de `main()`:

- `argc`: número de argumentos que se proporcionan (3 en el ejemplo)
- `argv`: array* con las cadenas que se proporcionan como argumentos

```
C:\>prueba cad1 cad2 cad3
```

Ejecuta `prueba.exe` con tres argumentos (cadenas de caracteres)

** Ya veremos qué son los arrays y cómo se manejan.*

Mercedes Gómez Albarrán

Funciones en C++: la función `main`

Comunicación con el sistema operativo: ¿qué devuelve `main`*?*

La función `main()` devuelve al sistema operativo un código de terminación de programa.

- `0`: *Todo OK*
- Distinto de `0`: *¡Ha habido un error!*

```
...

return 0; // Fin del programa
}
```

Mercedes Gómez Albarrán

Programas con subprogramas en C++

¿Qué subprogramas hay en el programa?

¿En qué punto del archivo .cpp se colocan? ¿Antes de `main()`, después de `main()`?

El compilador necesita saber qué subprogramas se han declarado para saber si son correctas las llamadas a los mismos:

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

Mercedes Gómez Albarrán

Programas con subprogramas en C++

¿Antes de `main()`, después de `main()`?

```
#include <iostream>
using namespace std;
int sumatorio (int N)  {
   int resultado = 0;
   for(int i = 1; i <= N; i++) resultado = resultado + i;
   return resultado;
}
int main()  {
  int num;
  do {
    cout << "Numero final: ";
    cin >> num;
  } while (num <= 0);
  cout << "La suma de los numeros entre 1 y " << num << " es: "
       << sumatorio (num) << endl;
return 0;
}
```

Mercedes Gómez Albarrán

Programas con subprogramas en C++

¿Antes de `main()`, después de `main()`? Los prototipos

Si la implementación de los subprogramas se coloca detrás de main, han de incluirse los prototipos de los subprogramas al principio del archivo

El prototipo es la cabecera del subprograma terminada en `;`

```
int comparacion(int val1, int val2);
void intercambia(double& valor1, double& valor2);
```

También se pueden omitir los nombres de los parámetros en el prototipo:

```
int comparacion(int, int);
void intercambia(double&, double&);
```

Los prototipos son fundamentales cuando se separa el programa en módulos (compilación separada).

Programas con subprogramas en C++

sumatorio.cpp

```
#include <iostream>
using namespace std;
int sumatorio (int);
int main()  {
  int num;
  do {
    cout << "Numero final: ";
    cin >> num;
  } while (num <= 0);
  cout << "La suma de los numeros entre 1 y " << num << " es: "
       << sumatorio (num) << endl;
  return 0;
}
int sumatorio (int N)  {
   int resultado = 0;
   for(int i = 1; i <= N; i++) resultado = resultado + i;
   return resultado;
}
```


Programas con subprogramas en C++

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

void divisores(int valor);
int sumatorio (int N);

int main()  {
  int num;

  do {
    cout << "Valor: ";  cin >> num;
  } while (num <= 0);
  cout << "Sumatorio: "  << sumatorio (num) << endl;
  cout << "Divisores: ";
  divisores(num);

  return 0;
}
```

Continúa...

Mercedes Gómez Albarrán

Programas con subprogramas en C++

```
// implementacion de las funciones

int sumatorio (int N)  {
   // codigo de la funcion
}

void divisores(int valor) {
  // codigo del procedimiento
}
```

¡¡ Las funciones pueden estar implementadas en el mismo orden que aparecen los prototipos o en orden cambiado !!

Mercedes Gómez Albarrán

Volvemos con el diseño descendente

Mercedes Gómez Albarrán

Volvemos con el diseño descendente

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

// prototipos
void espacioEnBlanco();
void escribeHOLA();
void escribeANA();
void escribeH();
void escribeO();
void escribeL();
void escribeA();
void escribeN();
```

```
// main
int main()  {

  escribeHOLA();
  espacioEnBlanco();
  escribeANA();
  return 0;
}

// implementacion de las funciones
void escribeH()
{
  cout << "*   *" << endl;
  cout << "*   *" << endl;
  cout << "*****" << endl;
  cout << "*   *" << endl;
  cout << "*   *" << endl;
}
```

Continúa...

Volvemos con el diseño descendente

```
void escribeO()
{  // codigo  }

void escribeL()
{  // codigo  }

void escribeA()
{  // codigo  }

void escribeN()
{  // codigo  }

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

```
void escribeHOLA()
{
   escribeH();
   escribeO();
   escribeL();
   escribeA();
}

void escribeANA()
{
   escribeA();
   escribeN();
   escribeA();
}
```

Continúa...

Mercedes Gómez Albarrán

Extras del tema

Mercedes Gómez Albarrán

Datos globales y datos locales a funciones

Datos en los programas

- Datos globales: declarados fuera de las funciones del programa. Existen durante toda la ejecución del programa.
- Datos locales a funciones: declarados en el cuerpo de alguna función. Existen sólo durante la ejecución de la función.

Ámbito y visibilidad de los datos

- Ámbito de los datos globales: todo el programa (desde su declaración). Se conocen dentro de las funciones.
- Ámbito de los datos locales a función: cuerpo de la función en la que se declaran.
  No se conocen fuera de esa función.
- Visibilidad de los datos: Un dato local que se llame igual que un dato global oculta ese dato global dentro de su función.

Mercedes Gómez Albarrán

Datos globales y datos locales a funciones

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

const int MAX = 100;
double ingresos;

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

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

Datos globales

Datos locales a `func()`

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)

Mercedes Gómez Albarrán

Datos globales y datos locales a funciones

Sobre el uso de datos globales en las funciones

No deben usarse datos globales en las funciones.

El uso de datos globales → riesgo de sufrir *efectos laterales*: modificaciones de esos datos que afecten de forma imprevista a otras partes de código.

Si se necesita un dato global, se definirá un parámetro en la función. El dato global se pasará como argumento en la llamada.

Como excepción, se pueden declarar constantes globales que contengan diversos parámetros globales del programa.

Mercedes Gómez Albarrán

Argumentos implícitos

Valores predeterminados para los parámetros (por valor)

```
void func(int i = 1);
```

`func(12);` `i` toma el argumento explícito `12`

`func();` `i` toma el argumento implícito (`1`)

Todos los parámetros que se declaren con argumentos implícitos deben encontrarse al final de la lista de parámetros:

```
void f(int i, int j = 2, int k = 3); // CORRECTO
void f(int i = 1, int j, int k = 3); // INCORRECTO
```

Los argumentos implícitos se declaran en el prototipo o en la cabecera de la función, pero NO en ambos. ¡Mejor en el prototipo!

Mercedes Gómez Albarrán

Argumentos implícitos

```
void f(int i, int j = 2, int k = 3);
...
  f(13); // i toma 13, j y k sus valores implícitos
  f(5, 7); // i toma 5, j toma 7 y k su valor implícito
  f(3, 9, 12); // i toma 3, j toma 9 y k toma 12
```

No podemos dejar `j` con el argumento por defecto y concretar `k`.

Mercedes Gómez Albarrán

Sobrecarga de funciones

Igual nombre, distintos parámetros

Podemos tener varias funciones con igual nombre pero con distinta lista de parámetros.

```
int abs(int n);
double abs(double n);


f(13); // argumento int --> primera función
f(-2.3); // argumento double --> segunda función
```

Mercedes Gómez Albarrán

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.

En http://es.creativecommons.org/ y http://creativecommons.org/ puedes saber más de Creative Commons.

Mercedes Gómez Albarrán