

Proyecto de Programación

Guía básica de estilo de programación

Escuela Politécnica Superior
Universidad Autónoma de Madrid

- **Organización de un programa**
 - Estructura de un programa
 - Cohesión de un módulo
 - Acoplamiento entre módulos
 - Estructura de un fichero de cabecera
 - Estructura de un fichero de código fuente
- **Nombrado de elementos de un programa**
 - Nombrado de constantes
 - Nombrado de variables
 - Nombrado de estructuras de datos
 - Nombrado de funciones
- **Codificación de un programa**
 - Codificación en general
 - Codificación de bloques de sentencias
 - Codificación de funciones
- **Documentación de un programa**
 - Documentación de un fichero
 - Documentación de una estructura de datos
 - Documentación de una función
 - Comentarios sobre bloques de sentencias
 - Comentarios sobre sentencias individuales

- **Organización de un programa**
 - Estructura de un programa
 - Cohesión de un módulo
 - Acoplamiento entre módulos
 - Estructura de un fichero de cabecera
 - Estructura de un fichero de código fuente
- **Nombrado de elementos de un programa**
 - Nombrado de constantes
 - Nombrado de variables
 - Nombrado de estructuras de datos
 - Nombrado de funciones
- **Codificación de un programa**
 - Codificación en general
 - Codificación de bloques de sentencias
 - Codificación de funciones
- **Documentación de un programa**
 - Documentación de un fichero
 - Documentación de una estructura de datos
 - Documentación de una función
 - Comentarios sobre bloques de sentencias
 - Comentarios sobre sentencias individuales

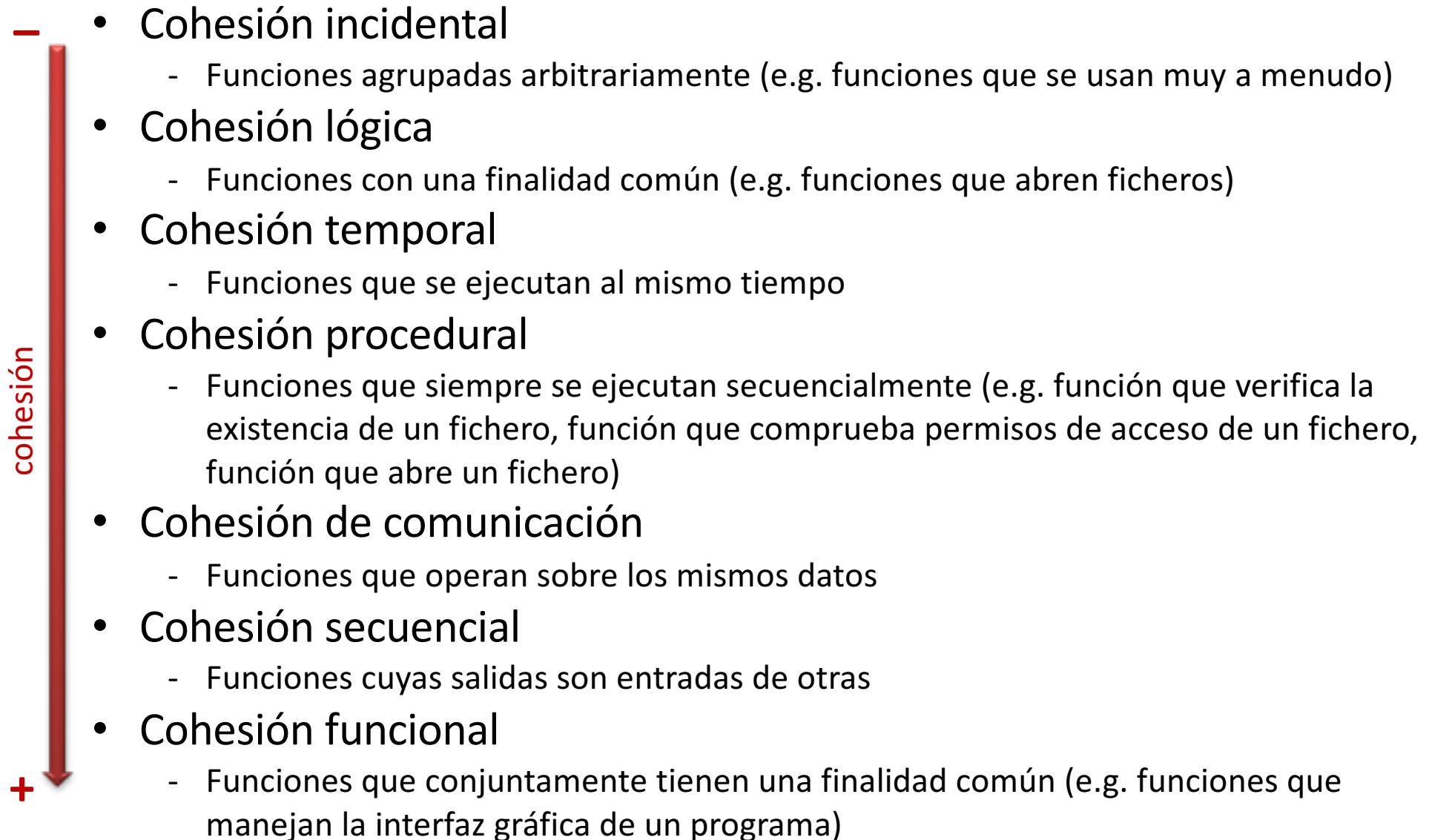
- **Estructura de un programa**

- En general, el código de un programa está dividido en **módulos**
 - e.g. módulo de interfaz gráfica, módulo interprete de comandos, módulo de lectura/escritura de datos en disco, etc.
- Un módulo es un conjunto de funciones que dentro del programa tienen algún tipo de relación (*cohesión*)
- Un módulo puede estar compuesto por uno o varios **ficheros** de código fuente (.c) y ficheros de cabecera (.h)
 - e.g. un módulo intérprete de comandos podría estar compuesto de interprete_comandos.c, interprete_comandos.h, carga_comandos.c, carga_comandos.h, procesado_cadenas.c, procesado_cadenas.h, ...
- Un módulo suele tener asignado un **subdirectorio** dentro del directorio del programa en el que se guardan sus ficheros de forma independiente
 - e.g. \proyecto_pprog\codigo\comandos, \proyecto_pprog\codigo\interfaz, etc.

Organización de un programa

- Estructura de un programa C
 - Archivos de cabecera **.h**
 - Macros: **#define**
 - Definición de tipos de datos: **typedef**
 - Declaración de funciones **públicas**
 - Archivos fuente **.c**
 - Definición de funciones **públicas**
 - Definición de funciones **privadas**

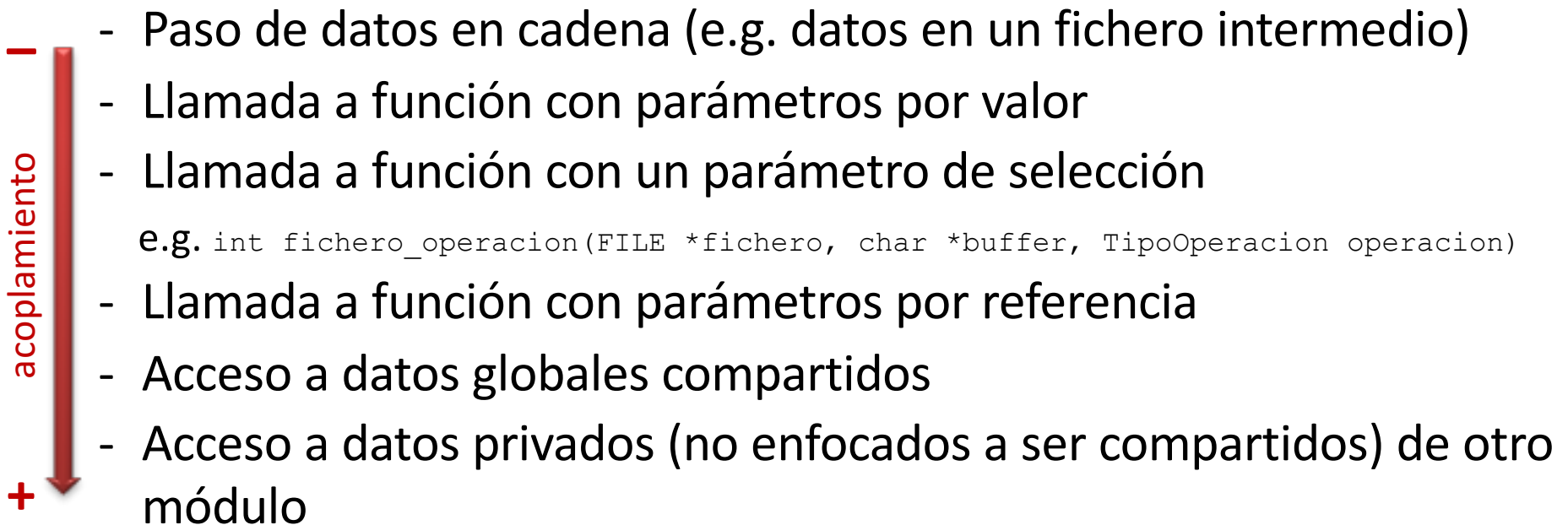
- **Cohesión de un módulo** (mayor cohesión mejor)

- 
- Cohesión incidental
 - Funciones agrupadas arbitrariamente (e.g. funciones que se usan muy a menudo)
 - Cohesión lógica
 - Funciones con una finalidad común (e.g. funciones que abren ficheros)
 - Cohesión temporal
 - Funciones que se ejecutan al mismo tiempo
 - Cohesión procedural
 - Funciones que siempre se ejecutan secuencialmente (e.g. función que verifica la existencia de un fichero, función que comprueba permisos de acceso de un fichero, función que abre un fichero)
 - Cohesión de comunicación
 - Funciones que operan sobre los mismos datos
 - Cohesión secuencial
 - Funciones cuyas salidas son entradas de otras
 - Cohesión funcional
 - Funciones que conjuntamente tienen una finalidad común (e.g. funciones que manejan la interfaz gráfica de un programa)

- **Acoplamiento entre módulos**

- El acoplamiento es una medida de la interacción entre dos módulos
 - cada interacción entre dos módulos es una potencial fuente de problemas, en cuanto que 1) modificaciones en uno pueden implicar modificaciones en otro, 2) demostrar la corrección de uno de forma aislada puede ser complicada

- Niveles de acoplamiento (menor acoplamiento mejor)

- 
- Paso de datos en cadena (e.g. datos en un fichero intermedio)
 - Llamada a función con parámetros por valor
 - Llamada a función con un parámetro de selección
e.g. `int fichero_operacion(FILE *fichero, char *buffer, TipoOperacion operacion)`
 - Llamada a función con parámetros por referencia
 - Acceso a datos globales compartidos
 - Acceso a datos privados (no enfocados a ser compartidos) de otro módulo

Organización de un programa

7

- Estructura de un fichero de cabecera (.h)

```
#ifndef LIBRO_H
#define LIBRO_H

/** Definición de valores constantes */
#define ERR 0
#define OK 1

/** Definición de estructuras de datos */
typedef struct {
    char *nombre;
    char *apellido;
} Autor;

typedef struct {
    char *titulo;
    Autor *autor;
    int anio;
} Libro;

/** Declaración de funciones publicas */
Libro *libroCrear(char *tituloLibro, int anioLibro, char *nombreAutor, char *apellidoAutor);

int libroLiberar(Libro *libro);

#endif
```


- Estructura de un fichero de código fuente (.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "libro.h"

/** Definicion de estructuras de datos y variables privadas */
typedef struct {
    Libro *libros;
    int numLibros;
} ListaLibros;

static ListaLibros librosCreados;

/** Declaracion de funciones privadas */
int libroGuardar(Libro *libro);

/** Definicion de funciones publicas */
Libro *libroCrear(char *tituloLibro, int anioLibro, char *nombreAutor, char *apellidoAutor) {
    // Aquí el codigo de la funcion...
}

int libroLiberar(Libro *libro) {
    // Aquí el codigo de la funcion...
}

/** Definicion de funciones privadas */
int libroGuardar(Libro *libro) {
    // Aquí el codigo de la funcion...
}
```

- **Organización de un programa**
 - Estructura de un programa
 - Cohesión de un módulo
 - Acoplamiento entre módulos
 - Estructura de un fichero de cabecera
 - Estructura de un fichero de código fuente
- **Nombrado de elementos de un programa**
 - Nombrado de constantes
 - Nombrado de variables
 - Nombrado de estructuras de datos
 - Nombrado de funciones
- **Codificación de un programa**
 - Codificación en general
 - Codificación de bloques de sentencias
 - Codificación de funciones
- **Documentación de un programa**
 - Documentación de un fichero
 - Documentación de una estructura de datos
 - Documentación de una función
 - Comentarios sobre bloques de sentencias
 - Comentarios sobre sentencias individuales

- **Una constante...**

- En general, su nombre posee todos sus caracteres en **mayúsculas** y si tiene un nombre compuesto, sus términos se separan con **guiones bajos**

MAX_WIDTH

- En C, su declaración es aconsejable con `const`, aunque también se suele hacer con macros `#define`

`#define MAX_WIDTH 256`  `const int MAX_WIDTH = 256;`

- En C, si se usa con el fin de declarar un conjunto finito de valores relacionados, es aconsejable hacerlo mediante **enumeraciones** en vez de macros

`#define LARGE 1`

`#define MEDIUM 2`

`#define SMALL 3`



`typedef CoffeeSize enum {`

`LARGE, MEDIUM, SMALL`

`};`

- **El nombre de una variable...**
 - desde el punto de vista semántico:
 - debería ser un **sustantivo** que explique en una o dos palabras el significado de la variable
 - debería pertenecer al **vocabulario** asociado al dominio de aplicación
 - desde el punto de vista sintáctico:
 - debería ser **corto** y en **singular** (o en plural si corresponde)
 - es aconsejable en **inglés**, aunque en español también es aceptable
 - en general, **empieza por letra minúscula**
 - si es compuesto, suele usar el **guion bajo** como separador, o seguir la notación *CamelCase* (la habitual en Java)
`drink_size, drinkSize`
 - debe evitar **ambigüedades**:
`Coffee coffee;`
`int num_coffees, numcoffees;`
`int num0, numO, num1, numl, num5, numS, num2, numZ, numn, numh;`

- El nombre de los nuevos tipos de datos...
 - Suelen ser **como los de las variables** pero **comenzando por mayúscula**
- El nombre de una estructura de datos...
 - Son **como los de los nuevos tipos** de datos, pero **precedidos de un guión bajo**

```
typedef struct _CoffeeCharacteristics {  
    char *acidity;  
    char *aroma;  
    char *body;  
    char *taste;  
} CoffeeCharacteristics;
```

```
typedef struct _Coffee {  
    char *name;  
    char *country;  
    CoffeeCharacteristics  
characteristics;  
} Coffee;
```

- El nombre de una función...
 - desde el punto de vista semántico
 - debería ser un **verbo más complemento** que expliquen en dos o tres palabras la acción o finalidad de la función

```
void print_coffee(Coffee coffee);  
double get_coffee_price(Coffee coffee, CoffeeSize size);
```
 - debería pertenecer al **vocabulario** asociado al dominio de aplicación
 - desde el punto de vista sintáctico
 - sigue las mismas convenciones que el nombre de una variable

- **Organización de un programa**
 - Estructura de un programa
 - Cohesión de un módulo
 - Acoplamiento entre módulos
 - Estructura de un fichero de cabecera
 - Estructura de un fichero de código fuente
- **Nombrado de elementos de un programa**
 - Nombrado de constantes
 - Nombrado de variables
 - Nombrado de estructuras de datos
 - Nombrado de funciones
- **Codificación de un programa**
 - Codificación en general
 - Codificación de bloques de sentencias
 - Codificación de funciones
- **Documentación de un programa**
 - Documentación de un fichero
 - Documentación de una estructura de datos
 - Documentación de una función
 - Comentarios sobre bloques de sentencias
 - Comentarios sobre sentencias individuales

- **Codificación en general**

- Consistente

- Usar un **mismo estilo de codificación**: idioma, tipo de nombres de variables/funciones, tipo de indentación, tipo de documentación, etc.

- Correcta

- Comprobar posibles **errores de ejecución**, e.g. divisiones por 0, bucles infinitos
 - Comprobar **retornos devueltos por llamadas a función**: e.g. comprobar que el FILE * devuelto por fopen no es NULL
 - Liberar **recursos**: e.g. llamar a free por cada llamada previa a malloc, llamar a fclose por cada llamada previa a fopen

- Extensible

- **Modularizar las funciones**
 - **Abstraer los datos**, e.g. usar tipos abstractos de datos, evitar “números mágicos” (i.e., valores constantes sin macros o variables const)

- **Codificación de bloques de sentencias**

- Bloques

- Deben ir debidamente “**indentados**”, i.e., con tabulaciones
 - Deben estar **separados** de otros bloques mediante al menos una línea en blanco
 - En C, si son condicionales (`if`, `switch`) o bucles (`for`, `while`), deben ir **delimitados** con llaves, aún cuando estén vacíos

```
while ( numCoffees < MAX_COFFEES ) {  
    printf("Coffee size: 1) small, 2) medium, 3) large\n");  
    scanf("%d", &coffeeSize);  
  
    coffees[numCoffees] = coffeeCreate(coffeType, coffeeSize);  
    if ( coffees[numCoffees] == NULL ) {  
        break;  
    }  
    numCoffees++;  
}
```

- **Codificación de funciones**

- **Funciones**

- En general, deberían ser de **longitud corta** (siempre menor de 150 líneas)
 - Se hace necesario dividir un problema en subproblemas
- En general, deberían tener **pocos argumentos de entrada**
- Deberán tener una **estructura sencilla**
 - Una función más larga de lo habitual, pero sencilla y bien estructurada, puede ser fácil de comprender
 - Una función compleja debería caber en una pantalla o en una hoja impresa para una más fácil comprensión
- Deberán tener sus **bloques de sentencias** bien separados
- Deberán siempre comprobar sus **argumentos de entrada**
- Deberán liberar/cerrar los **recursos** (memoria, ficheros, etc.) que han reservado/abierto antes de que acabe su ejecución (por flujo correcto o incorrecto)

- **Organización de un programa**
 - Estructura de un programa
 - Cohesión de un módulo
 - Acoplamiento entre módulos
 - Estructura de un fichero de cabecera
 - Estructura de un fichero de código fuente
- **Nombrado de elementos de un programa**
 - Nombrado de constantes
 - Nombrado de variables
 - Nombrado de estructuras de datos
 - Nombrado de funciones
- **Codificación de un programa**
 - Codificación en general
 - Codificación de bloques de sentencias
 - Codificación de funciones
- **Documentación de un programa**
 - Documentación de un fichero
 - Documentación de una estructura de datos
 - Documentación de una función
 - Comentarios sobre bloques de sentencias
 - Comentarios sobre sentencias individuales

- **Documentación de un fichero**

- Todo fichero debe ir precedido de una cabecera (claramente delimitada) en la que, entre otros aspectos, se especifiquen:
 - nombre, descripción (finalidad de las funciones que posee), autor, fecha, versión, histórico de revisiones

```
/* =====  
File:      coffee.c  
Version: 2.0  
Date:      Jan. 15, 2015  
Author:    Instructors  
  
Description:  
    Contains the implementation of the functions associated to the  
    Coffee data type.  
  
Revision history:  
    Jan. 1, 2014  Version 1.0 (initial release)  
    Jan. 15, 2014 Version 2.0  
        Added the functions that manage the coffee prices.  
===== */
```

- **Documentación de una estructura de datos**

- Toda estructura de datos debería ir precedida de un comentario clarificando su finalidad
- Los campos de una estructura suelen ir también acompañados de breves comentarios

```
/** The Coffee structure stores information of the different coffees
    managed by the sytem */
typedef struct {
    char *name;          /* Name of the coffee */
    char *country;       /* Origin country of the coffee */
    CoffeeCharacteristics characteristics; /* Main characteristics of
                                           the coffee (aroma, taste...) */
} Coffee;
```

- **Documentación de una función**

- Toda función debe ir precedida de una cabecera (claramente delimitada) en la que, entre otros aspectos, se especifiquen:
 - nombre, descripción, autor, fecha, versión, argumentos de entrada, retorno de salida

```
/* -----  
Function: get_coffee_price  
Date:     Jan. 15, 2015  
Author:   Instructors  
  
Description:  
    Computes the price of a particular coffee based on its type and  
    size.  
  
Input:  
    Coffee coffee: the coffee whose price is computed  
    CoffeeSize size: the size of the input coffee  
Output:  
    double: the price in Euro of the input coffee  
----- */
```

- **Comentarios sobre bloques de sentencias**
 - Consisten en una o varias líneas de comentario en la parte superior (antes de la primera sentencia) del bloque clarificando su significado

```
/* We compute the price of a coffee: we first set the base price of its  
   type, and then apply a price multiplier associated to its size */
```

```
double price = 0;
```

```
if ( strcmp(coffee.name, COFFEE_COLOMBIA) == 0 ) {  
    price = COFFEE_PRICE_COLOMBIA;
```

```
}
```

```
else {
```

```
    price = COFFEE_PRICE_BASIC;
```

```
}
```

```
switch (coffeeSize) {
```

```
    case SMALL:    price *= COFFEE_PRICE_MULT_S; break;
```

```
    case MEDIUM:  price *= COFFEE_PRICE_MULT_M; break;
```

```
    case LARGE:    price *= COFFEE_PRICE_MULT_L; break;
```

```
}
```

- **Comentarios sobre sentencias individuales**

- Consisten en una o dos líneas de comentario al lado (a la derecha) de una sentencia clarificando su significado

```
if ( *ptr = '#' ) { /* Removes comments at the end of the line */
    *ptr = '\0';
}
```

- En general, sólo deben usarse con sentencias complejas, de interpretación no trivial

- e.g. comentarios como el siguiente deben evitarse

```
numMaxLines = 100; /*Set the maximum number of lines to 100 */
```

- Se suelen usar para comentar el significado de sus variables en su declaración

```
int eof; /* flag; 1: we have reached the end of a file
          0: there are one or more lines to read */
```