Sistemes Operatius: Guia d'Estil

Introducció

Avui en dia, en el món de la programació, és molt important generar codi que sigui fàcilment llegible i comprensible tan per un mateix com per a la resta de les persones.

Un codi ben estructurat, seguint una sèrie de normes prèviament acordades, pot facilitar tan a l'autor del codi com als possibles lectors, la comprensió dels elements continguts en el mateix. D'aquesta forma es facilita la reutilització i manteniment del codi ja escrit.

L'objectiu d'aquest document és marcar una normativa que faciliti, mitjançant una sèrie de normes, la comprensió del codi que s'escriu, tan als alumnes com als professors.

Cal tenir en compte que aquesta guia d'estil no és un estàndard del llenguatge C, sinó que es tracta d'un conjunt de normes seleccionades a partir de les recomanacions fetes per diferents fonts.

Cal dir que el fet d'utilitzar correctament aquesta guia d'estil facilita la introducció de l'usuari en el mon de la programació real, ja que dota a l'usuari d'una sèrie de normes comunes, però no assegura a l'usuari el coneixement complet de com programar en C, ja que de guies d'estil ni han moltes i molt diverses.

1. Variables

1.1. Nomenclatura de les variables

La nomenclatura que donem a una variable és molt important, ja que ens facilita la lectura i comprensió del codi escrit. Per aquest motiu tindrem les següents normes:

a. Nom de les variables auto-explicatiu.

Això significa que el nom que prengui una variable ha de representar l'element o elements que contindrà. Aquest no ha de se excessivament llarg ni abreviat, sinó que haurem de trobar un punt entremig que satisfaci les nostres necessitats.

Exemple de males declaracions:

```
int n;                                  //El nom de variable massa curt.
int contador_de_preus_de_lector;        //El nom de variable massa llarg.
int n_comp_conns;                       //Abreviacions ambigües.
int dada;                               //Pot significar qualsevol cosa.
int cntdr;                              //Eliminar lletres.
int conneccio_wgc;                      //Només tu saps què vol dir wgc.
```

Exemple de bones declaracions:

```
int num_errors;                         //Num és una abreviació estàndard.
int conneccio_dns;                      //Molta gent sap què vol dir DNS.
int operand1;                           //Molt útil si existeix operand2.
char nom_fitxer[MAXF];                  //És el nom d'un fitxer!
```

Excepcions:

Poden donar-se excepcions en el cas de variables puntuals de tipus auxiliar que només tenen sentit en una part molt petita del nostre codi.

```
int i, j, k;                            //Comptador d'un bucle.
int aux;                                //Auxiliar per fer un swap.
```

b. Nom de les variables escrit en minúscules i separat per _.

Exemple de males declaracions:

```
int coneccio_DNS;                       //No pot contenir majúscules.
int numErrors;                          //Separar paraules amb _.
int Data;                               //No pot començar en majúscula.
```

1.2. Declaració de variables

La ubicació i el format de la declaració de les variables també és molt important, ja que si aquestes estan ben ubicades, podrem fer-nos una idea de la magnitud de dades a controlar dins el procediment o funció.

a. Variables declarades al principi

Totes les variables hauran d'estar declarades al principi del procediment o funció. No hi podran haver declaracions de variables barrejades amb sentències del nostre programa.

Exemple de males declaracions:

```
int dia;
char lletra1;

dia = 1;
lletra1 = 'c';
int i;                                  //Variable mal declarada!
for (i = 0; i<10; i++) {
      ...
```

La variable i no està declarada al principi i per tant pot confondre.

b. No utilitzar variables globals

La utilització de variables globals de forma genèrica és una mala pràctica en el món de la programació, ja que fàcilment podem trobar-nos en que no sabem on s'utilitza aquella variable, on no i quin valor té actualment.

Nota: Aquesta norma no es tindrà en compte al llarg de la pràctica 1.

c. Valors de coma flotant

Durant la declaració i utilització de valors de coma flotant caldrà mostrar sempre almenys un nombre abans i després de la coma flotant, per així evitar errors, tan de comprensió, com de programació.

Exemple de males declaracions:

```
float resultat = 5;                     //Cal declarar resultat com 5.0.
int valor1 = 3;

resultat = valor1 / 2;                  //Sense 2.0, la divisió és entera!
      ...
```

1.3. Consells

a. Variables declarades en una sola línia

Es recomana que totes les variables que tenen algun tipus de relació entre si es declarin en una mateixa línia. L'objectiu d'aquesta pràctica és agrupar en blocs totes aquelles variables que molt probablement s'utilitzin alhora.

Exemple:

```
int i, j, k;                            //Comptadors.
int dia = 1, mes = 1, any = 1900;       //Variables de data.
char nom[MAXNOM], cognoms[MAXCOGNOM];  //Variables de nom d'usuari.
char codi_lletra;                       //Altres variables
int num_usuaris;
```

2. Constants

2.1. Nomenclatura de les constants

La nomenclatura que donem a una constant és molt important, ja que ens facilita la lectura i comprensió del codi escrit. Per aquest motiu tindrem les següents normes:

a. Nom de les constants auto-explicatiu.

Això significa que el nom que prengui una constant ha de representar l'element o elements que contindrà. Aquest no ha de se excessivament llarg ni abreviat, sinó que haurem de trobar un punt entremig que satisfaci les nostres necessitats.

Exemple de males declaracions:

```
#define N 10                            //Nom de constant massa curt.
#define MAXIM_NUMERO_DE_USUARIS 100     //Nom de constant massa llarg.
#define M_CON_EST 50                    //Abreviacions ambigües.
#define MAX 10                          //Pot significar qualsevol cosa.
#define MX_CNXNS 50                     //Eliminar lletres.
#define MAX_WGC 6                       //Només tu saps què vol dir wgc.
```

Exemple de bones declaracions:

```
#define MAX_CONNEXIONS 10               //MAX és una abreviació estàndard.
#define MY_IP "192.168.0.13"            //Molta gent sap què vol dir IP.
#define MAX_ARRAY 100                   //Molt útil per definir arrays.
#define FILE_CONF "configuració.txt"    //És el nom d'un fitxer!
```

b. Nom de les constants escrit en majúscules i separat per _.

Exemple de males declaracions:

```
#define max_CONNEXIONS 10          //No pot contenir minúscules.
#define MAXCONNEXIONS 10               //Separar paraules amb _.
```

2.2. Declaració de constants

La ubicació de la declaració de les constants és molt important, ja que si aquestes no estan ben ubicades, poden induir a errors o dificultar la lectura del programa.

a. Constants entre llibreries i funcions

Les constants hauran d'estar declarades després d'incloure totes les llibreries que inclourà el nostre programa o mòdul i abans de la declaració de procediments i funcions.

Exemple de males declaracions:

```
#include <stdio.h>
#define MAX 10
int contarMAX();
#include <stdlib.h>
#define N 15
```

Les constants estan repartides i barrejades amb els llibreries i funcions de forma que es dificulta la lectura i comprensió del codi.

Exemple de bones declaracions:

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

#define MAX 10
#define N 15

int contarMAX();
```

2.3. Consells

a. Definició de constants per cadenes i arrays

Es recomana que, per definir la mida de les cadenes de caràcters i arrays, s'utilitzin constants, d'aquesta forma, si en algun moment volem modificar la mida d'aquesta cadena o array, només ens caldrà modificar el valor de la constant, sense haver de retocar tot el codi.

Exemple:

```
#define MIDA_ARRAY 100

int main () {
      int i;
      int el_meu_array[MIDA_ARRAY];

      for (i=0; i<MIDA_ARRAY; i++) {
            ...
      }
```

D'aquesta forma, si en algun moment volem modificar la mida de l'array, només ens cal modificar el valor de la constant MIDA_ARRAY, enlloc de tocar la declaració i la condició del bucle *for*.

b. Definició de constants per valors amb significat

Es recomana que, en el cas de trobar-nos amb un programa que treballa amb valors que tenen un significat especial (ex: Codis de colors d'una llibreria gràfica), es

defineixin constants, amb noms que el programador entengui, que prenguin aquests valors.

Exemple:

```
#define NEGRE       0
#define BLAU_FOSC   1
#define VERD_FOSC   2
#define CYAN        3
#define VERMELL     4
#define ROSA        5
#define TARONJA     6
#define GRIS_CLAR   7
 ...


int resetColor (int color) {
      color = NEGRE;                  //Millor que "color = 0;"
      return color;
}
```

D'aquesta forma, s'entén molt més si diem que color val NEGRE, que si diem que color val 0.

3. Procediments i funcions

3.1. Nomenclatura dels procediments i funcions

La nomenclatura que donem a un procediment o a una funció és molt important, ja que ens facilita la lectura i comprensió del codi escrit. Per aquest motiu tindrem les següents normes:

a. Nom dels procediments i funcions auto-explicatiu.

Això significa que el nom que prengui un procediment o una funció ha de representar, el més acurat possible, allò que fa. El nom no ha de se excessivament llarg ni abreviat, sinó que haurem de trobar un punt entremig que satisfaci les nostres necessitats.

Exemple de males declaracions:

```
int f();                                //Nom de funció massa curt.
void sumarTotsElsElementsDeLArray();    //Nom de procediment massa llarg.
int contCon();                          //Abreviacions ambigües.
int funcio();                           //Pot significar qualsevol cosa.
void cntrCslls();                       //Eliminar lletres.
int ghx2();                             //Només tu saps què vol dir ghx2.
char darthVader();                      //No té sentit!
```

Exemple de bones declaracions:

```
int trobarMax();                        //MAX és una abreviació estàndard.
void  canviarIp();                      //Molta gent sap què vol dir IP.
int contarElements();                   //Deixa clar què fa la funció.
void escriureMenu();
```

b. Nom dels procediments i funcions escrit en "lowerCamelCase".

"lowerCamelCase" és una convenció de noms que es caracteritza per escriure una sèrie de paraules unides entre si (sense deixar espais en blanc entre paraules). Per tal que les diferents paraules es puguin distingir, la primera lletra de cada paraula estarà escrita en majúscules. La particularitat que té "lowerCamelCase" és que la primera paraula anirà escrita en minúscules.

Exemple de males declaracions:

```
void lamevafuncio();                    //Falta majúscules principi paraula.
void LaMevaFuncio();                    //Primera lletra minúscula.
void LAMEVAFUNCIO();                    //No pot anar tot en majúscules.
void la_Meva_Funcio();                  //No poden haver _.
```

Exemple de bones declaracions:

```
void laMevaFuncio();                    //Compleix lowerCamelCase.
```

3.2. Mida dels procediments i funcions

Per tal que els procediments i funcions compleixin amb la seva funció dins del mon de la programació, cal delimitar la mida que poden prendre aquests:

a. Mida màxima dels procediments i funcions.

Tot procediment o funció que creem, no podrà excedir les 60 línies de codi.
El motiu principal per aquesta decisió, és degut a que 60 línies de codi caben a la majoria de monitors, de forma que el programador que llegeix aquest procediment o funció podrà veure tot el contingut d'aquest, en la seva totalitat, sense la necessitat d'haver de moure el text amunt i avall.

En casos excepcionals es poden acceptar procediments o funcions que, per la seva pròpia naturalesa, no es poden reduir a menys de 60 línies de codi.

4. Tipus

4.1. Nomenclatura dels tipus propis

La nomenclatura que donem a un tipus propi és molt important, ja que ens facilita la lectura i comprensió del codi escrit. Per aquest motiu tindrem les següents normes:

a. Nom dels tipus propis auto-explicatius.

Això significa que el nom que prengui un tipus propi ha de representar, el més acurat possible, allò que contindrà. El nom no ha de se excessivament llarg ni abreviat, sinó que haurem de trobar un punt entremig que satisfaci les nostres necessitats.

Exemple de males declaracions:

```
typedef struct {
        …
} T;                                    //No és una abreviació comprensible.

typedef struct {
        …
} ConjuntDeDadesQueTeUnUsuari;          //És massa llarg.

typedef struct {
        …
} Jug;                                  //Abreviacions ambigües.
```

Exemple de bones declaracions:

```
typedef struct {
        …
} Jugador;                              //Defineix què contindrà.

typedef struct {
        …
} JocOca;
```

b. Nom dels tipus propis escrit en "UpperCamelCase".

"UpperCamelCase" és una convenció de noms que es caracteritza per escriure una sèrie de paraules unides entre si (sense deixar espais en blanc entre paraules). Per tal que les diferents paraules es puguin distingir, la primera lletra de cada paraula estarà

escrita en majúscules. La particularitat que té "UpperCamelCase" és que la primera paraula anirà escrita en majúscules.

Exemple de males declaracions:

```
typedef struct {
      ...
} elmeutipus;                          //Falta majúscules principi paraula.

typedef struct {
      ...
} elMeuTipus;                          //Primera lletra majúscula.

typedef struct {
      ...
} ELMEUTIPUS;                          //No pot anar tot en majúscules.

typedef struct {
      ...
} El_Meu_Tipus;                        //No poden haver-hi _.
```

Exemple de bones declaracions:

```
typedef struct {
      ...
} ElMeuTipus;                          //Compleix UpperCamelCase.
```

5. Fitxers

5.1. Estructura d'un fitxer .c amb "main"

L'estructura d'un fitxer .c que contingui el procediment main serà una mica diferent de la resta, tal i com podrem veure més endavant.

a. Llibreries

En primer lloc trobarem totes les llibreries necessàries per tal que el nostre programa funcioni correctament. Aquestes hauran d'estar ordenades de major a menor importància, de forma que en primer lloc trobarem les llibreries del sistema (stdio.h, stdlib.h, ...), en segon lloc trobarem les llibreries pròpies del nostre sistema (conio.h, Windows.h, ...) i finalment trobarem els mòduls que haguem creat nosaltres per aquests projecte.

Exemple Includes:

```
//Llibreries del sistema
#include <stdio.h>
#include <stdlib.h>

//Llibreries del nostre sistema
#include <conio.h>
#include <Windows.h>

//Llibreries pròpies
#include "my_library.h"
```

b. Constants

Un cop definides totes les llibreries que utilitzarem, passarem a definir les constants que utilitzarà el nostre procediment principal (main) i que no trobarem definides en altres llibreries o mòduls.

c. Tipus propis

En tercer lloc passarem a definir els nostres tipus propis. Es recomana que es defineixin mòduls específics per tal de definir els tipus propis del nostre programa.

d. Procediment principal

Finalment passarem a escriure el procediment principal. Aquest ha de ser l'únic procediment o funció del fitxer, i ha de ser el mes curt i concís possible per tal de facilitar la comprensió global del programa.

Exemple de fitxer .c amb "main":

```
//Llibreries del sistema
#include <stdio.h>
#include <stdlib.h>

//Llibreries del nostre sistema
#include <conio.h>
#include <Windows.h>

//Llibreries pròpies
#include "la_meva_llibreria.h"

//Constants
#define CONSTANT1  1
#define CONSTANT2  2

//Tipus propis
typedef struct {
      int camp1;
      char camp2;
} ElMeuTipus;

//Procediment principal
int main (int argc, char* argv[]) {
      ...
}
```

5.2. Estructura d'un fitxer .c de mòdul

L'estructura d'un fitxer .c que formi part d'un mòdul (conjunt de fitxer .c i fitxer .h) serà la següent:

a. Include a fitxer .h

El primer que trobarem sempre en un fitxer .c que formi part d'un mòdul és un include al seu fitxer .h corresponent. No afegirem mai cap include més en un fitxer .c, ja que aquests els trobarem en el fitxer .h del mateix mòdul.

b. Procediments i funcions

A continuació trobarem la implementació de tots els procediments i funcions del mòdul. En aquest fitxer no trobarem ni definició de constants, ni definició de tipus, no cap altre element que no siguin els especificats fins ara.

Exemple de fitxer .c:

```
//Include del fitxer .h del mateix mòdul
#include "el_meu_modul.h"

//Procediments i funcions
int funcio1() {
	....
}

int funcio2() {
	....
}

int funcio3() {
	....
}
```

5.3. Estructura d'un fitxer .h

L'estructura d'un fitxer .h serà sempre la següent:

a. "Define Guards"

El primer que hem d'escriure son els anomenats "define guards". Els "define guards" son una eina, a nivell de compilador, que tenim per tal d'evitar que el nostre compilador entri en un bucle infinit de compilació i d'altres problemes. Al principi del nostre fitxer .h sempre afegirem les línies:

```
//Define Guard
#ifndef _EL_MEU_MODUL_H
#define _EL_MEU_MODUL_H

//El nostre codi

#endif
```

On _EL_MEU_MODUL_H serà el nom del nostre fitxer .h començant amb el caràcter _, escrivint totes les lletres en majúscules i separant les paraules amb un _, ja que estem definint una constant i cal respectar el format de les constants.

Cal afegir al final de tot la sentència #endif per tal de tancar el condicional a nivell de compilador.

b. Llibreries

En segon lloc trobarem totes les llibreries necessàries per tal que el nostre programa funcioni correctament. Aquestes hauran d'estar ordenades de major a menor importància, de forma que en primer lloc trobarem les llibreries del sistema (stdio.h, stdlib.h, ...), en segon lloc trobarem les llibreries pròpies del nostre sistema (conio.h, Windows.h, ...) i finalment trobarem els mòduls que haguem creat nosaltres per aquests projecte.

Exemple Includes:

```
//Llibreries del sistema
#include <stdio.h>
#include <stdlib.h>

//Llibreries del nostre sistema
#include <conio.h>
#include <Windows.h>

//Llibreries pròpies
#include "my_library.h"
```

c. Constants

Un cop definides totes les llibreries que utilitzarem, passarem a definir les constants que utilitzaran els procediments i funcions del nostre mòdul.

d. Tipus propis

A continuació passarem a definir els nostres tipus propis.

e. Capçaleres de procediments i funcions

Finalment escriurem les capçaleres de tots els procediments i funcions que implementi el nostre mòdul.

Exemple de fitxer .h:

```
//Define Guards
#ifndef _EL_MEU_MODUL_H
#define  _EL_MEU_MODUL_H

//Llibreries del sistema
#include <stdio.h>
#include <stdlib.h>

//Llibreries del nostre sistema
#include <conio.h>
#include <Windows.h>

//Llibreries pròpies
#include "la_meva_llibreria.h"

//Constants
#define CONSTANT1  1
#define CONSTANT2  2

//Tipus propis
typedef struct {
      int camp1;
      char camp2;
} ElMeuTipus;

//Procedimens i funcions
int funcio1();
int funcio2();
int funcio3();

#endif
```

5.4. Consells

a. Nomenclatura dels fitxers

Tot i que el nom dels fitxers pot ser el que el programador decideixi, recomanem que segueixi el següent format.

- Tots els noms del fitxers, tan el .c com els .h, s'haurien d'escriure en minúscules separats per _.
- El fitxer que contingui el procediment principal s'hauria d'anomenar main.c per tal de facilitar la seva distinció d'entre la resta de fitxers .c pertanyents a mòduls.

b. Fitxers .h i llibreries

Tot i que en un fitxer .h no cal afegir les llibreries que ja estan incloses en un altre fitxer .h inclòs en el primer, recomanem que tot fitxer .h contingui tots els includes pertinents a totes les llibreries que utilitzi.

Exemple:
Si el fitxer A.h inclou al mòdul B.h i C.h, i el mòdul B.h també inclou al mòdul C.h només caldria escriure dins del fitxer A.h:

```
#include "B.h"
```

Ja que de forma transitiva, el fitxer A.h inclou a C.h a través de B.h.

De totes formes el que es recomana és que igualment s'escrigui:

```
#include "B.h"
#include "C.h"
```

Per tal de facilitar la lectura i comprensió del nostre programa.

6. Espaiat

Si volem que el codi que escrivim sigui fàcilment llegible i amable a la vista necessitarem definir un espaiat comú que faciliti aquesta llegibilitat sense afegir grans espais en blanc que dificultin la lectura.

6.1. Espaiat vertical

Per espaiat vertical entenem el conjunt de línies en blanc que deixarem per tal de fer el nostre codi més llegible.

a. Espaiat de "paràgrafs"

En tot codi de programació, com en tot escrit, es poden distingir diferents paràgrafs. Els paràgrafs son conjunts de frases que tenen un sentit propi com a conjunt, i que per tant s'han de llegir alhora. D'aquesta forma en la programació també existeixen paràgrafs. Aquests paràgrafs són conjunts de sentències que tendim a agrupar (definició de constants, definició de tipus, bucles, ...).

Per tal de fer el nostre codi més llegible deixarem sempre una línia en blanc per separar aquests paràgrafs.

Exemple espaiat vertical:

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

#define CONSTANT 1

int main () {
	int  enter1;
	int enter2;
	char caracter1;

	for (enter1 = 0; enter1 < enter2; enter1++) {
		....
	}

	if (caracter1 == '3') {
		...
	}
}
```

6.2. Espaiat horitzontal

Per espaiat horitzontal entenem el conjunt de caràcters en blanc que deixarem per tal de fer el nostre codi més llegible.

a. Operadors unaris

Mai deixarem un espai en blanc per separar un operador unari de l'expressió amb la que treballa.

Exemple mal espaiat horitzontal:

```
scanf("%d", & a);          //L'operador & i la variable a han d'estar junts.
int * p;                   //L'operador * i la variable p han d'estar junts.
i ++;                      //L'operador ++ i la variable i han d'estar junts.
```

b. Operadors binaris

Sempre deixarem un espai en blanc per separar un operador binari de les expressions amb les que treballa.

Exemple mal espaiat horitzontal:

```
a=b+c;                     // Falten espais al voltant dels operadors  = i +.
if (a==b) {                // Falten espais al voltant de l'operador ==.
if (a<b) {                 // Falten espais al voltant de l'operador <.
```

Exemple bon espaiat horitzontal:

```
a = b + c;
if (a == b) {
if (a < b) {
```

Excepció: L'operador -> mai anirà espaiat.

c. Comes i punts i comes

Sempre deixarem un espai en blanc després de les comes i punts i comes.

Exemple mal espaiat horitzontal:

```
int a,b,c;                      //Falten espais després de les comes.
void func(int a,int b);         //Falten espais després de les comes.
for (a = 0;a < b;a++) {         //Falten espais després dels punts i comes.
```

d. Parèntesis

Sempre deixarem un espai en blanc fora dels parèntesis, tan d'obertura com de tancament, però mai a l'interior dels mateixos.

Exemple mal espaiat horitzontal:

```
if(a == b){                    //Falten espais abans de ( i després de ).
if ( a == b ) {                //Sobren espais després de ( i abans de ).
```

e. Claus

Sempre deixarem un espai en blanc abans de la obertura de claus.

Exemple mal espaiat horitzontal:

```
if (a == b){                   //No hi ha espai abans de {
```

f. Case-Switch

Mai deixarem un espai en blanc abans dels : d'un "case" dins d'una sentència switch.

Exemple mal espaiat horitzontal:

```
case 'A' :                     //No hi ha espai abans de :
```

7. Indentació

Per tal que el codi que escrivim sigui fàcilment llegible i amable a la vista necessitarem definir un sistema d'indentació comú que ens faciliti la lectura dels diferents blocs.

7.1. Indentació de blocs

Per a cada un dels diferents blocs del nostre programa (entenem per bloc nou tota obertura de clau, case d'un switch, ...) aplicarem un nivell d'indentació, de forma que els nous blocs quedin tabulats a la dreta i es pugui seguir amb facilitat el recorregut del programa. En el moment de tancar els blocs, les claus aniran a la mateixa altura que la sentència que les ha obert, i els breaks els trobarem a la mateixa altura que les sentencies de dins els cases.

Exemple mala indentació:

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

      #define CONSTANT  1             //Les constants no son un bloc nou.

int main () {
int  enter1;                          //Cal indentar totes les variables
int enter2;
char caracter1;

      if (condició) {
      sentencia1;                     //Cal indentar la sentència.
                  }                   //La clau va a l'altura de l'if.

            switch (expresio) {       //No cal indentar, està a nivell 1.
            case 1:
            sentencia2;               //Cal indentar la sentencia.
                  break;
            case 2:
                  sentencia3;
            break;                    //Cal indentar el break.
      }
}
```

Exemple bona indentació:

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

#define CONSTANT 1

int main () {
	int  enter1;
	int enter2;
	char caracter1;

	if (condició) {
		sentencia1;
	}

	switch (expresio) {
		case 1:
			sentencia2;
			break;
		case 2:
			sentencia3;
			break;
	}
}
```

7.2. Consells

Recomanem que cada Indentació equivalgui a un tabulador o a quatre espais en blanc.

8. Comentaris

Els comentaris són una part molt important del nostre codi. Tot i que aquests no afecten a l'execució del nostre programa, faciliten enormement la comprensió del codi escrit.

8.1. Comentaris de fitxers

Per tal d'entendre correctament l'objectiu d'un fitxer d'un mòdul, cal afegir una capçalera explicativa. Aquesta capçalera s'haurà d'afegir al principi de tots els fitxers i contindrà el propòsit del fitxer (perquè s'ha creat aquest fitxer?), l'autor o autors del fitxer, la data de creació del fitxer i la data de la última modificació del fitxer.

Capçalera comentaris de fitxers:

```
/*********************************************
*
* @Proposit:
* @Autor/s:
* @Data creacio:
* @Data ultima modificacio:
*
*********************************************/
```

8.2. Comentaris de funcions

Per tal d'entendre correctament l'objectiu d'un procediment o funció, cal afegir una capçalera explicativa. Aquesta capçalera s'haurà d'afegir a sobre de cada procediment o funció i contindrà la finalitat de la funció (què fa la funcio?), els diferents paràmetres de la funció (de que serveix el paràmetre, és d'entrada/sortida, ...) i el que retorna la funció ("----" si es un procediment):

Capçalera comentaris de procediments i funcions:

```
/*********************************************
*
* @Finalitat:
* @Parametres:
* @Retorn:
*
*********************************************/
```

Exemple 1 comentaris de procediments i funcions:

```
/*********************************************
*
* @Finalitat: Suma els valors dels paràmetres introduïts.
* @Parametres:     in: numero1 = primer operand de l'operació de suma.
*                  in: numero2 = segon operand de l'operació de suma.
* @Retorn: Retorna la suma de numero1 + numero2.
*
*********************************************/
int sumar (int numero1, int numero2);
```

Exemple 2 comentaris de procediments i funcions:

```
/*********************************************
*
* @Finalitat: Crea una cua amb fantasma o no i la deixa buida.
* @Parametres:     in/out: la_cua = cua no definida que passara a
*                         guardar la cua buida.
*                  in: fantasma = si val 0 es creara sense fantasma. Per
*                         qualsevol altrevalor, es crearà amb fantasma.
* @Retorn: ----.
*
*********************************************/
void crearCua (cua *la_cua, int fantasma);
```

8.3. Comentaris de línia

Per tal d'entendre correctament algunes parts del nostre codi caldrà afegir comentaris enmig del nostre codi. Els comentaris del nostre codi han de ser intel·ligents i explicatius, i han de servir perquè els possibles lectors del nostre codi entengui les parts complexes d'aquest. No ens servirà de res comentar obvietats o fer explicacions confuses i poc clares.

Exemple de mals comentaris:

```
//Incremento el valor de 'x' en 1.
x = x +1;

//Faig un bucle fins MAX_ELEM.
for (i = 0; i < MAX_ELEM; i ++) {
      if (max < arr[i]) {
            max = arr[i];
      }
}
```

Exemple de bons comentaris:

```
//Bucle que busca el valor maxim dins l'array arr.
for (i = 0; i < MAX_ELEM; i ++) {
      if (max < arr[i]) {
            max = arr[i];
      }
}
```

a. Comentaris laterals

Si es decideix que es volen situar els comentaris a continuació del codi, en la mateixa línia, caldrà tabular aquest comentari i tots els comentaris que segueixin aquest criteri a la mateixa altura.

Exemple de comentaris mal tabulats:

```
int numero; //Guarda el valor del numero introduit per l'usuari.
char nom_usuari[MAX_NOM]; //Nom de l'usuari que introdueix el valor.
int edat;    //Edat de l'usuari.
```

Exemple de comentaris ben tabulats

```
int numero;                //Guarda el valor del numero introduit per
                           //l'usuari.
char nom_usuari[MAX_NOM]; //Nom de l'usuari que introdueix el valor.
int edat;                  //Edat de l'usuari.
```