Laboratori de Fonaments de Programació

Jordi Blasco Planesas

2019-10-31

Contents

Chapter 1

Introducció

El present recurs és un petit exemple de la documentació que es pot generar amb **R** i la llibreria *bookdown*, en la qual s'exposen FAQS recopilades del **Laboratori de Fonaments de Programació**.

La documentació utilitza les següents icones per referenciar els blocs:

 Indica que el codi mostrat està en **llenguatge algorísmic**.

 Indica que el codi mostrat està en **llenguatge C**.

 Mostra l'execució d'un programa en **llenguatge C**.

Chapter 2

PAC01

2.1 Llenguatge algorísmic

El llenguatge algorísmic l'hem d'entendre com una aproximació al món real, el qual utilitza unes normes definides per nosaltres mateixos. En aquest punt encara no parlem de programes escrits en **C**, en **Java**, en **Python** o en **PHP**, per dir alguns llenguatges de programació.

Per exemple, en el llenguatge algorísmic que utilitzem a l'assignatura definim un bloc de variables de la següent forma:

```
var
    edat: integer;
    pes: real;
end var
```

Que es tracti d'un llenguatge més proper al món real no significa que no s'hagin de complir unes determinades regles. Com es pot veure en aquest exemple, una d'aquestes regles és que quan definim variables ho precedim amb `var` i ho finalitzem amb `end var`.

Hem decidit utilitzar aquesta forma de llenguatge algorísmic, tot i que també ho podríem haver plantejat de la següent forma :

```
variable
    enter edat
    decimal pes
fvariable
```

Remarcar que aquest segon exemple **és incorrecte**, no segueix la nomenclatura del llenguatge algorísmic definit a l'assignatura. El correcte és el primer exemple.

El llenguatge algorísmic és com fer una aproximació formal a la realitat, no és un llenguatge de programació en sí com és **C**, **Java** o similars. Per tant no és un llenguatge que es pugui compilar i executar amb l'IDE utilitzat a l'assignatura, el qual està preparat únicament per interpretar i executar codi programat en llenguatge C.

Ara bé la gran pregunta: i per què és necessari primer dissenyar l'algorisme, si puc directament programar-ho en C?

Un algorisme ens permet dissenyar un programa sense tenir presents les particularitats de cada llenguatge de programació. Aquesta aproximació formal a la realitat dels algorismes ens faciliten poder fer posteriorment una traducció ràpida a qualsevol llenguatge de programació simplement coneixent les equivalències corresponents. Per exemple, el primer cas si el programem en C equival a:

```
int edat;
float pes;
```

El codi en C no el podem canviar, ja que si en comptes de posar int utilitzem enter, el compilador de C no comprèn el mot i ens donarà un error de codi.

Si mai hem programat és normal que aquest plantejament sobti al principi, però és important que poc a poc es vagi veient les diferències entre llenguatge algorísmic i llenguatge C.

2.2 Llenguatge algorísmic vs llenguatge C

En general:

- **Llenguatge algorísmic**: proper al llenguatge natural, es tracta d'una convenció que adoptem nosaltres mateixos per definir el un programa formalment. Els algorismes tenen una sèrie de normes i sentències que nosaltres definim (**Nomenclàtor**), però que no són de cap forma interpretables per un ordinador. Per tant un algorisme **no pot ser compilat ni executat**.
- **Llenguatge C**: es tracta d'un llenguatge de programació que sí comprèn un ordinador. Això significa que únicament podem utilitzar les seves comandes i les seves normes per tal que el codi pugui ser compilat i executat sense problemes.

El llenguatge algorísmic és un pseudocodi que ens ajuda a definir com funciona un programa. No està lligat a cap llenguatge de programació, amb el que les accions que realitzarà, la forma de definir variables, etc. és genèrica. Funcions com `writeString()`, `readInteger()` o `writeChar()` formen part del llenguatge algorísmic: indiquen una acció genèrica a realitzar, com és escriure una cadena de caracters, llegir un enter o escriure un caràcter. Quan es vulgui codificar aquest algorisme en un llenguatge de programació concret com és C, només caldrà saber les comandes pròpies de C que ens permeten implementar l'algorisme.

La programació en C funciona exclusivament amb la sintaxi definida per aquest llenguatge de programació. Instruccions com `scanf()` i `printf()` són pròpies de C.

A mode d'exemple:

Algorisme: volem introduïr la lectura de la llum de casa nostra; una possible implementació és:

```
var
    lecturaMensual: integer;
end var

writeString("Introdueix la lectura mensual de la llum (kWh): ");
lecturaMensual := readInteger();
```

Llenguatge C: en aquest llenguatge no existeixen les funcions algorísmiques `writeString()` ni `readInteger()`, però en canvi sí que tenim vàries funcions pròpies de C que ens permeten llegir un valor per teclat i assignar-lo a una variable d'entorn. Per tant, les accions algorísmiques anteriors correspondran a la següent codificació en C:

```c
int lecturaMensual;

printf("Introdueix la lectura mensual de la llum (kWh): ");
scanf("%d", &lecturaMensual);
```

És molt important que es vegi clarament què és un algorisme i què és un programa en C.

2.3 Impressió de valors incorrecta

Quan es mostra per pantalla el contingut d'alguna variable amb `printf()`, és important eliminar el prefix `&` de la variable:

```c
#include <stdio.h>

int main(int argc, char **argv){

    int idAvio;

    printf("Introdueix l'identificador d'avió : ");
    scanf("%d", &idAvio);

    printf(">> Has escollit l'avió amb id %d \n", &idAvio);

    return 0;
}
```

El resultat de l'execució és:

```
Introdueix l'identificador d'avió : 9
>> Has escollit l'avió amb id -1078693464
```

Quan fem referència a `&idAvio` estem obtenint realment la posició de memòria on resideix la variable `idAvio`. Per obtenir el seu valor cal eliminar de dins `printf()` el prefix `&` de la variable `idAvio`:

```c
#include <stdio.h>

int main(int argc, char **argv){
```

```
    int idAvio;

    printf("Introdueix l'identificador d'avió : ");
    scanf("%d", &idAvio);

    printf(">> Has escollit l'avió amb id %d \n", &idAvio);

    return 0;
}
```

La sortida generada ara sí és correcta:

```
Introdueix l'identificador d'avió : 9
>> Has escollit l'avió amb id 9
```

2.4 Com definir un enumeratiu

La definició d'un tipus enumeratiu es fa de la següent forma:

```
type
    typeName = {VALUE1, VALUE2, VALUE3, ... , VALUEn};
end type
```

Els elements `VALUE1`, `VALUE2`, `VALUE3`... acaben sent constants, i el valor que de cadascun és:

```
VALUE1 = 0
VALUE2 = 1
VALUE3 = 2
{ ... }
VALUEn = n-1
```

Posteriorment no és possible fer un canvi de valor d'aquests elements de tipus enumeratiu.

2.5 Com utilitzar un enumeratiu

Una enumeració simplement és una assignació d'un valor enter, començant pel 0 i incrementant-se en 1, a la sèrie d'elements que l'hi has definit.

Per exemple, podem tenir la següent definició:

```
typedef enum {MALE, FEMALE} tGender;
```

Això significa que internament `MALE == 0` i `FEMALE == 1`. Si l'ordre de la definició l'haguessis fet al revés, `{FEMALE, MALE}`, tindríem que `FEMALE == 0` i `MALE == 1`.

D'aquesta forma es llegeix un enter i es compara el seu valor amb un o altre element de l'`enum` per tal de fer l'acció que desitgem. Una possible implementació en llenguatge C seria:

```c
#include <stdio.h>

int main(int argc, char **argv) {

   typedef enum {MALE, FEMALE} tGender;

   tGender gender;

   printf("Type patient gender: 0 for MALE, 1 for FEMALE\n");
   scanf("%d", &gender);

   if (gender == MALE) {
      printf("Patient gender MALE\n");
   } else {
      if (gender == FEMALE) {
         printf("Patient gender FEMALE\n");
      } else {
         printf("Incorrect option\n");
      }
   }
}
```

2.6 Especificador d'un enumeratiu

Els enumeratius en llenguatge C, `enum`, utilitzen l'especificador `%u` independentment del tipus que s'hagi definit per l'enumeratiu.

Exemple:

```c
#include <stdio.h>

typedef enum {PRIVAT, PUBLIC} tTransport;

int main(int argc, char **argv) {
    tTransport tipusTransport;

    printf("Com et desplaces a la feina (0=privat, 1=públic) ? : ");
    scanf("%u", &tipusTransport);
    printf("Et desplaces a la feina amb transport (0=privat, 1=públic) : ");
```

```
    printf("%u\n", tipusTransport);
}
```

2.7 Lectura de caràcters en C

En el llenguatge C la lectura d'un `char` pot comportar-se de forma inadequada si prèviament el buffer d'entrada conté algun caràcter previ.

Imaginem que volem crear un programa molt senzill que donat un número de DNI i la seva lletra, ens concateni els dos valors i ho mostri per pantalla. Una possible forma d'implementar aquest programa en C seria:

```c
#include <stdio.h>

int main(int argc, char **argv) {

    int dniNum;   /* número del DNI */
    char dniChar; /* lletra del DNI */

    printf("Introdueix el número del DNI: ");
    scanf("%d", &dniNum);
    printf("Introdueix la lletra del DNI: ");
    scanf("%c", &dniChar);

    printf("\nEl DNI introduit és: %d-%c\n", dniNum, dniChar);

    return 0;
}
```

Què passa si executem aquest codi? Que veiem que es comporta de forma incorrecta, ja que no ens arriba a demanar la lletra del DNI, mostrant directament el resultat:

```
Introdueix el número del DNI: 12345678
Introdueix la lletra del DNI:
El DNI introduit és: 12345678-
```

Quan teclegem el primer enter el que fem realment és introduir un número + un `intro` al final de tot. El número queda assignat a la variable `dni_num`, i l'`intro` és llegit com un caràcter i s'assigna a la variable `dni_char`. Per aquest motiu C interpreta que les dues variables ja tenen valor i finalitza el programa.

Com podem solucionar aquest comportament? Buidant l'`intro` del buffer d'entrada abans de llegir el caràcter, i això ho podem fer amb la comanda `getchar()`. Aquesta comanda llegeix un caràcter del buffer d'entrada i el buida del buffer.

Per tant es pot corregir el programa anterior de la següent forma:

```c
#include <stdio.h>

int main(int argc, char **argv) {

    int dni_num; // número del DNI
    char dni_char; // lletra del DNI

    printf("Introdueix el número del DNI: ");
    scanf("%d", &dni_num);
    getchar();
    printf("Introdueix la lletra del DNI: ");
    scanf("%c", &dni_char);

    printf("\nEl DNI introduit és: %d-%c\n", dni_num, dni_char);

    return 0;
}
```

Si ara executem, ja funcionarà com desitgem:

```
Introdueix el número del DNI: 12345678
Introdueix la lletra del DNI: B

El DNI introduit és: 12345678-B
```

En cas de necessitat, amb `getChar()` es pot guardar el caràcter del buffer en una variable, per tal de tractar-lo posteriorment:

```c
char nomVariable;
nomVariable = getChar();
```

2.8 Lectura de float en C

El separador de valors decimals (tipus `float`) en C **és el punt**, no la coma. Per aquest motiu quan s'introdueix un valor decimal des de teclat, sempre ho farem amb un punt:

Exemple:

```c
#include <stdio.h>

int main(int argc, char **argv) {}

    /* Variable que contindrà el pes d'una persona */
    float pes;
```

```c
    /* Lectura de la dada per teclat (el separador de decimals és un . ) */
    printf("Introdueix el pes (kg) d'una persona : ");
    scanf("%f", &pes);

   /* Es mostra el valor decimal per pantalla */
   printf("Has introduit el pes = %.1f kg.\n", pes);

   return 0;
}
```

L'execució serà:

```
Introdueix el pes (kg) d'una persona : 79.440
Has introduit el pes = 79.4 kg.
```

Chapter 3

PAC02

3.1 define vs const

La definició de constants tant es pot fer amb `define` com amb `const`. Tot i això, la forma de comportar-se d'aquestes dues opcions és completament diferent, si ve el resultat final és el mateix:

- `define`: quan utilitzem aquesta opció no es desa en cap posició de memòria el valor de la constant. El que es fa realment és que en els passos previs a la pròpia compilació del programa, el preprocessador substitueix totes les referencies del define pel valor indicat.

Per exemple, si tenim el següent programa amb una constant creada amb `define` :

```c
#include <stdio.h>
#define MIDA 8

char lletres[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
int vertical, horitzontal;

int main(int argc, char **argv) {
    /* font: https://en.wikipedia.org/wiki/Chess */

    for (vertical=MIDA; vertical>=1; vertical--) {
        for (horitzontal=0; horitzontal<=MIDA-1; horitzontal++) {
            printf("%c%d ", lletres[horitzontal], vertical);
        }
        printf("\n");
    }
}
```

Abans de la compilació, el preprocessador entre altres accions elimina comentaris i substitueix totes les referències `MIDA` per `8`:

```c
#include <stdio.h>

char lletres[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
int vertical, horitzontal;

int main(int argc, char **argv) {

    for (vertical=8; vertical>=1; vertical--) {
        for (horitzontal=0; horitzontal<=8-1; horitzontal++) {
            printf("%c%d ", lletres[horitzontal], vertical);
        }
        printf("\n");
    }
}
```

Per tant la definició de constants amb `define` es comporta com si d'un "cercar-reemplaçar" d'un processador de textos es tractés. No es desa cap constant en memòria, però per contra, el programa ocuparà una mica més per la substitució directa de referències que fa; la substitució la fa en tot el programa, no es pot limitar a un àmbit concret (per exemple només dins d'una funció).

- `const`: en aquest cas sí que es reserva una posició de memòria. En C es comporta igual com si fos una variable, però la qual únicament funciona en mode lectura: no li podem modificar el valor.

A més, `const` ens permet també dir quin tipus de valor tindrà la constant: si és de tipus `float`, `int`, `char`… amb el que aquest fet ens dóna un punt addicional de control, ja que ens assegurem que el tipus de valor assignat serà el correcte pel programa.

Amb aquest tipus de definició de constant, l'exemple anterior quedaria de la següent forma:

```c
#include <stdio.h>
const int MIDA 8

char lletres[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
int vertical, horitzontal;

int main(int argc, char **argv) {

    /* La constant MIDA està desada en memòria */
    printf("posició en memòria de la constant MIDA : %p \n", &MIDA);

    for (vertical=MIDA; vertical>=1; vertical--) {
        for (horitzontal=0; horitzontal<=MIDA-1; horitzontal++) {
            printf("%c%d ", lletres[horitzontal], vertical);
        }
        printf("\n");
    }
}
```

Com es pot veure, és possible obtinir l'adreça en memòria on es desa la constant `MIDA`. En aquest cas, sí que pots definir una constant amb `const` i fer que només afecti un àmbit determinat (per exemple, que la constant estigui definida únicament dins d'una funció).

Aquestes són les principals diferències entre `define` i `const` a l'hora de definir una constant; `define` es va

crear molt abans que no la sentència `const`, amb el que és un habit força habitual decantar-se per aquesta opció per temes més històrics.

3.2 Precisió en variables float

Hi ha alguns valors decimals determinats que no es poden representar de forma precisa en una variable de tipus `float`. La millor solució pels casos que tractem és arrodonir al número de decimals que realment necessitem.

Si en canvi volem sí o sí treballar amb tots els decimals, podem optar per utilitzar un tipus de dada que tingui major precisió que `float`: `double`.

Per exemple, el següent programa retorna el resultat esperat si es desa en un `double`, i no així si es fa en un `float`:

```c
#include <stdio.h>

int main() {

    float num1;
    float num2;
    float resultat1;
    double resultat2;

    num1 = 1.3;
    num2 = 17;
    resultat1 = num1 + num2;
    printf("resultat amb float: %f\n", resultat1);
    resultat2 = num1 + num2;
    printf("resultat amb double: %f\n", resultat2);
    return 0;
}
```

La sortida que genera és:

```
resultat amb  float: 18.299999
resultat amb double: 18.300000
```

3.3 Expressions

3.3.1 Exemple 1: esParell

Imaginem que ens demanen un algorisme que indiqui si un número és parell.

Una possible solució seria:

```
algorithm esParell
    var
        numero: integer;
        isParell: boolean;
    end var

    writeString("Introdueix un número : ");
    numero:= readInteger();
    isParell:= (numero mod 2 = 0);

    writeString("El numero ");
    writeInteger(numero);
    writeString(" és parell? ");
    writeBoolean(isParell);

end algorithm
```

La variable `isParell` prendrà el valor `TRUE` si el número es parell i `FALSE` en cas contrari. No ha calgut utilitzar cap estructura `if-else` per resoldre l'algorisme.

Una possible forma de codificar-ho en llenguatge C és:

```c
#include <stdio.h>

typedef enum {FALSE, TRUE} boolean;

int main(int argc, char **argv) {

    int numero;
    boolean isParell;

    printf("Introdueix un número : ");
    scanf("%d", &numero);
    isParell = (numero % 2 == 0);

    printf("El número %d és parell? (0=FALSE, 1=TRUE) : %u \n", numero, isParell);

    return 0;
}
```

3.3.2 Exemple 2: capDeSetmana

Imaginem que volem fer un programa molt senzill que ens digui si avui és cap de setmana o no. El seu algorisme seria el següent:

```
type
    dies = {DILLUNS, DIMARTS, DIMECRES, DIJOUS, DIVENDRES, DISSABTE, DIUMENGE};
end type

algorithm capDeSetmana

    var
        esCapDeSetmana: boolean;
        diaSetmana: dies;
    end var

    writeString("Quin dia de la setmana és avui ?\n");
    writeString("Per DILLUNS tecleja 0\n");
    writeString("Per DIMARTS tecleja 1\n");
    writeString("Per DIMECRES tecleja 2\n");
    writeString("Per DIJOUS tecleja 3\n");
    writeString("Per DIVENDRES tecleja 4\n");
    writeString("Per DISSABTE tecleja 5\n");
    writeString("Per DIUMENGE tecleja 6\n");

    diaSetmana:= readInteger();
    esCapDeSetmana:= (diaSetmana = DISSABTE or diaSetmana = DIUMENGE);

    writeString("Avui és cap de setmana?");
    writeBool(esCapDeSetmana);

end algorithm
```

La variable `boolean esCapDeSetmana` prendrà el valor de `TRUE` o `FALSE` en funció del resultat d'avaluar l'expressió. No és necessari la utilització d'estructures condicionals `if-else` que veurem més endavant en el curs.

Una possible forma de codificar-ho en llenguatge C és:

```c
#include <stdio.h>

typedef enum {FALSE, TRUE} boolean;
typedef enum {DILLUNS, DIMARTS, DIMECRES, DIJOUS, DIVENDRES, DISSABTE, DIUMENGE} dies;

int main(int argc, char **argv) {

    boolean esCapDeSetmana;
    dies diaSetmana;

    printf("\nQuin dia de la setmana és avui ?\n");
    printf("Per DILLUNS tecleja 0\n");
    printf("Per DIMARTS tecleja 1\n");
    printf("Per DIMECRES tecleja 2\n");
    printf("Per DIJOUS tecleja 3\n");
    printf("Per DIVENDRES tecleja 4\n");
    printf("Per DISSABTE tecleja 5\n");
    printf("Per DIUMENGE tecleja 6\n");
```

```
    scanf("%u", &diaSetmana);
    esCapDeSetmana = (diaSetmana == DISSABTE || diaSetmana == DIUMENGE);

    printf("Avui és cap de setmana (0 == FALSE, 1 == TRUE) ? %u\n", esCapDeSetmana);

    return 0;
}
```

Varis punts a considerar:

- Recordem que en C no hi ha el tipus primitiu `boolean`, al mateix nivell que existeix un `int`, un `float` o un `char`… Per tant definirem sempre el tipus `boolean` com un `enum` amb els components `{FALSE, TRUE}`.
- Quan definim una variable de tipus `enum`, la llegirem/escriurem com a `%u`. Ho podríem fer com a `%d`, però us retornarà un warning tot i que el resultat sigui correcte. El tipus `%u` és igual que un enter `%d` però sense signe: això significa que amb `%d` podem tractar valors negatius com -12 i amb `%u` això no és possible, però com que sabem que els valors que pot prendre un `enum` sempre seran $>= 0$, ens convé utilitzar `%u`.
- Per les particularitats dels `boolean` en el llenguatge de programació C que ja hem comentat abans, l'entrada i sortida de valors d'un `boolean` serà numèrica. Per facilitar la comprensió podem mostrar per pantalla un literal que ens indiqui que 0 equival a `FALSE` i 1 a `TRUE`.
  - L'assignació de valors en els algorismes la fem amb `:=`, però en llenguatge C es fa amb `=`.
  - La comparació en els algorismes la fem amb `=`, però en llenguatge C es fa amb `==`.
  - L'operació lògica `or` dels algorismes es tradueix en C amb l'operand `||`.
  - L'operació lògica `and` dels algorismes es tradueix en C amb l'operand `&&`.

3.3.3 Exemple 3: ginTonicPreparation

Imaginem que volem preparar un gintònic. Sabem el volum de ginebra i de tònica que utilitzarem, i quina és la capacitat de la copa de baló que el contindrà.

Hem vist una oferta per internet i hem comprat glaçons metàl · lics d'acer inoxidable… però se'ns ha anat una mica el cap i n'hem comprat un total de 20 unitats.

Volem fer un programa que, utilitzant únicament expressions, ens digui si podem preparar o no el gintònic en funció del número de glaçons que li volem posar:

- si el nombre de glaçons caben dins de la copa, retornarà `true`.
- en cas contrari, retornarà `false`.

Per tant el que ha de fer el nostre programa bàsicament és validar si el volum de ginebra + tònica + (glaço) * número de glaçons supera o no el volum de la copa.

L'algorisme podria ser el següent:

```
const
    GIN: real = 50.0;              { in ml }
    TONIC: real = 200.0;           { in ml }
    GLASS: real = 620.0;           { in ml }
    METAL_ICE_CUBE: real = 42.875; { in ml }
end const

algorithm ginTonicPreparation
```

```
    var
        numMetalIceCubes: integer;
        isPossible: boolean;
    end var

    writeString("Number of metal ice cubes ? (integer) : ");
    numMetalIceCubes:= readInteger();

    isPossible:= (GLASS   (GIN + TONIC + METAL_ICE_CUBE * numMetalIceCubes));

    writeString("Can you make a gin & tonic? : ");
    writeBoolean(isPossible);

end algorithm
```

L'expressió que dóna valor a `isPossible` s'ocupa d'avaluar el volum de la copa respecte el resultant de ginebra, tònica i glaçons.

La seva traducció a C podria ser:

```c
#include <stdio.h>

#define GIN 50.0              /* in ml */
#define TONIC 200.0           /* in ml */
#define GLASS 620.0           /* in ml */
#define METAL_ICE_CUBE 42.875 /* in ml */

typedef enum {FALSE, TRUE} boolean;

int main(int argc, char **argv) {

    int numMetalIceCubes;
    boolean isPossible;

    printf("Number of metal ice cubes ? (integer) : ");
    scanf("%d", &numMetalIceCubes);

    isPossible = GLASS >= (GIN + TONIC + METAL_ICE_CUBE * numMetalIceCubes);

    printf("Can you make a gin & tonic? (0=FALSE, 1=TRUE) : %u\n", isPossible);

    return 0;
}
```

Per realitzar el càlcul en C també s'ha utilitzat una expressió.

3.3.4 Exemple 4: ginTonicFreeMl

Anem a evolucionar l'exemple anterior del gintònic: imaginem ara que volem que el nostre programa ens digui el volum (en mililitres) que queda lliure a la copa una vegada posat un determinat nombre de glaçons.

L'algorisme quedaria de la següent forma:

```
const
    GIN: real = 50.0;              { in ml }
    TONIC: real = 200.0;           { in ml }
    GLASS: real = 620.0;           { in ml }
    METAL_ICE_CUBE: real = 42.875; { in ml }
end const

algorithm ginTonicFreeMl

    var
        numMetalIceCubes: integer;
        volumeFree: real;
    end var

    writeString("Number of metal ice cubes ? (integer) : ");
    numMetalIceCubes:= readInteger();

    volumeFree:= GLASS - (GIN + TONIC + METAL_ICE_CUBE * numMetalIceCubes);

    writeString("How many free ml in the glass? : ");
    writeReal(volumeFree);

end algorithm
```

I la codificació en C :

```c
#include <stdio.h>

#define GIN 50.0              /* in ml */
#define TONIC 200.0           /* in ml */
#define GLASS 620.0           /* in ml */
#define METAL_ICE_CUBE 42.875 /* in ml */

typedef enum {FALSE, TRUE} boolean;

int main(int argc, char **argv) {

    int numMetalIceCubes;
    float volumeFree;

    printf("Number of metal ice cubes ? (integer) : ");
    scanf("%d", &numMetalIceCubes);

    volumeFree = GLASS - (GIN + TONIC + METAL_ICE_CUBE * numMetalIceCubes);

    printf("How many free ml in the glass ? : %.3f ml \n", volumeFree);
```

```
    return 0;
}
```

3.3.5 Exemple 5: scoutingBasquet

Imaginem que fem tasques d'scouting per les seccions de bàsquet femení i masculí del nostre club, i ens han encarregat cobrir alguna de les tres places següents:

- Per l'equip femení: una pivot que com a mínim faci 195cm d'alçada.
- Per l'equip femení: una base, l'alçada de la qual sigui inferior a 170cm.
- Per l'equip masculí: un base que sigui més alt de 175cm però a la vegada que no superi els 190cm.

El nostre programa demanarà per teclat si es tracta d'una jugadora o un jugador, i quina és la seva alçada. A continuació amb expressions avaluarà les condicions introduides i si les compleix per alguna de les tres places disponibles, l'escollirà (`isDrafted`).

Una possible forma de codificar en C aquest programa seria:

```c
#include <stdio.h>

typedef enum {FALSE, TRUE} boolean;
typedef enum {MALE, FEMALE} tGender;

int main(int argc, char **argv) {

    boolean isPointGuard; /* Point Guard = base */
    boolean isCenter;     /* Center = pivot */
    boolean isDrafted;
    int height;
    tGender gender;

    printf("Gender (0=MALE, 1=FEMALE) : ");
    scanf("%u", &gender);
    printf("Heigth (integer value) : ");
    scanf("%d", &height);

    /* Primer mirem si la posició de base femení o masculí la podem cobrir o no */
    isPointGuard =
        (height < 170 && (gender == FEMALE)) ||
        (height < 190 && height > 175 && (gender == MALE));

    /* A continuació comprovem si es tracta de la pívot femenina que busquem */
    isCenter = (height >= 195 && (gender == FEMALE));

    /* Només que es compleixi alguna de les dues expressions anteriors
       (que isPointGuard sigui TRUE o que isCenter sigui TRUE), el jugador/a
       serà escollit per formar part de les nostres seccions de bàsquet */
    isDrafted = isPointGuard || isCenter;

    printf("\nIs drafted (0=FALSE, 1=TRUE) ? : ");
```

```
    printf("%u\n", isDrafted);
}
```

Fixeu-vos que `isPointGuard` i `isCenter` són variables de tipus `boolean`, ja que l'avaluació de les expressions també serà de tipus `boolean`.

Hi poden haver altres codificacions igual de vàlides, aquesta no és la única solució possible.

Chapter 4

PAC03

4.1 Significat dels arguments del main

La principal diferència entre la definició `main(int argc, char **argv)` i `main()` és que la primera opció està preparada per rebre arguments quan s'executa el programa i no així la segona.

Per exemple, si tens el següent programa compilat en C i li passes una sèrie d'arguments des de la línia de comandes:

```
$> programa a1 a2 a3
```

Amb el main definit com a `main(int argc, char **argv)` pots accedir des de dins del programa a tots els arguments passats; així tindràs que:

```
argc = 4

argv[0] = "programa"
argv[1] = "a1"
argv[2] = "a2"
argv[3] = "a3"
```

El propi sistema operatiu s'ocupa de donar-li el valor a l'argument int `argc` (número total d'arguments inclòs el nom del programa), amb el que únicament t'has de preocupar de passar els arguments . D'altra banda, `argv` és un array de punters on cadascun d'ells apunta a un argument format per una cadena de caràcters; així argv contindrà a cadascuna de les seves posicions els arguments passats des de línia de comandes, i en la posició 0 el propi nom del programa.

Si en canvi tens definit el programa com a `main()`, simplement no tens forma d'accedir als arguments que li puguis arribar a passar. Hi ha moltes vegades que les dades les pots tenir ja definides dins del propi programa o les vagis a consultar a una font externa, amb el que no tenir la capacitat de processar arguments no suposa cap impediment a l'hora d'executar el teu programa.

4.2 Assignar valors a un vector

La lectura i assignació de valors a un vector es realitza de la següent forma en llenguatge algorísmic:

```
const
    MAX_TEMP: integer = 2;
end const

algorithm lecturaTemperatures

    var
        vTemperatures: vector[MAX_TEMP] of float;
    end var

    writeString("Introdueix la lectura 1 : ");
    vTemperatures[1] := readReal();

    writeString("Introdueix la lectura 2 : ");
    vTemperatures[2] := readReal();

    writeString("Els valors introduïts han estat : ");

    writeString("> Valor de la posició ");
    writeInteger(1);
    writeString(" : ");
    writeReal(vTemperatures[1]);

    writeString("> Valor de la posició ");
    writeInteger(2);
    writeString(" : ");
    writeReal(vTemperatures[2]);

end algorithm
```

I en llenguatge C:

```c
#include <stdio.h>

#define MAX_TEMP 2

int main(int argc, char **argv) {

    float vTemperatures[MAX_TEMP];
    int i;

    printf("Introdueix la lectura 1 : ");
    scanf("%f", &vTemperatures[0]);

    printf("Introdueix la lectura 2 : ");
    scanf("%f", &vTemperatures[1]);

    printf("> Valor de la posició %d : %.1f \n", 0, vTemperatures[0]);
    printf("> Valor de la posició %d : %.1f \n", 1, vTemperatures[1]);

    return 0;
```

```
}
```

Cal remarcar una diferència important:

- En **llenguatge algorísmic** les posicions del vector van des **de la 1 fins a la N**, sent N el número total d'elements del vector.
- En **llenguatge C**, van des **de la 0 fins a la N-1**, sent N el número total d'elements del vector.

4.3 Stack smashing detected

Aquest missatge d'error es produeix quan s'intenta accedir/operar amb una posició d'un vector que no l'hem definit prèviament. Es pot donar per diferents situacions que acaben generant el mateix problema.

- Cas 1: es defineix un vector de n-posicions, però en comptes de començar per la posició 0 ho fem per la 1. Això és incorrecte: recordeu que en C la posició inicial d'un vector sempre és la 0, i la final sempre és mida-1. Exemple, per un vector de 3 posicions tindrem:

```
int v[3];

v[0] = 10;  /* posició del vector vàlida */
v[1] = 13;  /* posició del vector vàlida */
v[2] = 24;  /* posició del vector vàlida */
v[3] = 0;   /* posició del vector no vàlida! */
```

- Cas 2: es defineix un vector amb menys posicions de les que necessitem. Per exemple, si tenim:

```
int v[2];
```

Significa que les posicions reservades en memòria per aquest vector són:

```
v[0] = 10;  /* posició del vector vàlida */
v[1] = 13;  /* posició del vector vàlida */
v[2] = 24;  /* posició del vector no vàlida! */
```

Per tant qualsevol operació amb `v[2]` ens generarà l'error indicat. Si volem que el vector contingui 3 elements només cal definir correctament la seva mida:

```
int v[3];
```

Chapter 5

PAC04

5.1 Com tractar elements d'un vector amb un bucle

Imaginem que ens demanen un programa que realitzi dues accions:

1. llegir des del canal estàndard d'entrada (teclat) 5 números i introduir-los en un vector d'enters.
2. mostrar pel canal estàndard de sortida (pantalla) els 5 números del vector d'enters del punt anterior.

L'algorisme podria ser el següent:

```
const
   MAX_NUMS: integer = 5;
end const

algorithm vectorDeNumeros

   var
      i: integer;
      vectorNumeros: vector[MAX_NUMS] of integer;
   end var

   { Assignar valor a cada posició del vector des de teclat }
   for i := 1 to MAX_NUMS do
      writeString("Introdueix número : ");
      vectorNumeros[i] := readInteger();
   end for

   { Mostrar per pantalla quin valor hi ha a cada posició del vector:}
   { es podria haver utilitzat un bucle for, però ho implemento amb un while }
   { perquè es vegi que també és possible fer-ho }
   i := 1;

   while i   MAX_NUMS do

      writeString("La posició ");
      writeInteger(i);
```

```
      writeString(" del vector conté el número ");
      writeInteger(vectorNumeros[i]);

      { És molt important que amb un bucle while incrementem la variable que utilitzem d'índex }
      { abans de finalitzar tot el bloc d'instruccions que executa, ja que en cas contrari el seu }
      { valor sempre seria de i == 1 }
      i := i+1;

   end while

end algorithm
```

Com es pot veure, per tal d'insertar/llegir els elements d'un vector aprofitem la iteració d'un bucle per recorre'ls tots, un a un, mitjançant una variable que utilitzem d'índex (en aquests casos, la variable `i`).

La traducció a C de l'algorisme podria ser així:

```c
#include <stdio.h>

#define MAX_NUMS 5

int main(int argc, char **argv) {

   int vectorNumeros [MAX_NUMS];
   int i;

   /* Assignar valor a cada posició del vector des de teclat */
   for (i = 0; i < MAX_NUMS; i++) {
      printf("Introdueix número : ");
      scanf("%d", &vectorNumeros[i]);
   }

   /* Mostrar per pantalla quin valor hi ha a cada posició del vector:
      es podria haver utilitzat un bucle for, però ho implemento amb un while
      perquè es vegi que també és possible fer-ho */
   i = 0;

   while (i < MAX_NUMS) {

      printf("\nLa posició %d del vector conté el número %d", i, vectorNumeros[i]);

      /* És molt important que amb un bucle while incrementem la variable que utilitzem d'índex
         abans de finalitzar tot el bloc d'instruccions que executa, ja que en cas contrari el seu
         valor sempre seria de i == 0 */
      i = i+1;
   }
}
```

5.2 Entrada continua de valors amb un bucle

Imaginem que hem de fer un programa que vagi demanant números indefinidament i que finalitzi únicament en el cas que el número introduit sigui parell.

Una possible forma de fer-ho utilitzant un únic while seria la següent:

```c
#include <stdio.h>

int main(int argc, char **argv) {

   int numero;

   /* Demanem una primera vegada el número a validar
      just abans d'entrar al bucle */
   printf("Tecleja un número parell : ");
   scanf("%d", &numero);

   while ((numero % 2) != 0) {

      /* Entra al bucle en el cas que el
         residu de la divisió per 2 sigui
         diferent de 0 (equival a no ser parell) */
      printf("El número %d no és parell !!\n", numero);

      /* Tornem a demanar un número, ara ja
         dins del bucle */
      printf("Tecleja un número parell : ");
      scanf("%d", &numero);
   }

   printf("El número %d és parell.\n", numero);
   return 0;
}
```

Abans d'entrar al bucle demanem un el valor de la variable numero. A continuació s'utilitza la condició de bucle per validar si es tracta d'un número parell o senar:

- Si el número es parell, no s'entra al bucle.
- Si el número és senar es compleix la condició del bucle i s'hi entra; dins del bucle es torna a demanar un valor per la variable numero i es torna a actuar igual que abans:
  - Si és senar, no se surt del bucle.
  - En cas contrari, se surt del bucle.

Finalment es mostra per pantalla el missatge "El número X és parell".

Chapter 6

PAC05

6.1 Exemple: nomines

Imaginem que volem un programa que ens permeti entrar les nòmines de tots els empleats de la nostra empresa. Un empleat el definim com a nom (cadena de caràcters) + nòmina (real). El programa ha de mostrar al final de tot la relació de nòmines de tots els empleats, la mitjana de totes les nòmines de l'empresa, i qui cobra més i menys a l'empresa.

Una possible forma de programa-ho seria la següent:

```c
#include <stdio.h>
#include <string.h>

#define MAX_EMPLEATS 5
#define MAX_NOM 20+1

typedef struct {
   char nom[MAX_NOM];
   float nomina;
} tEmpleat;

int main(int argc, char **argv) {

   tEmpleat vEmpleats[MAX_EMPLEATS];
   int i = 0;
   int maxNomina = 0;
   int minNomina = 0;
   float sumaNomines = 0;

   for (i=0; i<MAX_EMPLEATS; i++) {
      printf("\nNom empleat : ");
      scanf("%s", vEmpleats[i].nom);
      printf("Nòmina : ");
      scanf("%f", &vEmpleats[i].nomina);
   }
```

```
   printf("\nLlistat de nòmines d'empleats : \n\n");

   for (i=0; i<MAX_EMPLEATS; i++) {
      sumaNomines = sumaNomines + vEmpleats[i].nomina;
      if (vEmpleats[i].nomina > vEmpleats[maxNomina].nomina) {
         maxNomina = i;
      }
      if (vEmpleats[i].nomina < vEmpleats[minNomina].nomina) {
         minNomina = i;
      }
      printf("%s --> %.2f €\n", vEmpleats[i].nom, vEmpleats[i].nomina);
   }

   printf("\nMitjana nònimes : %.2f €", sumaNomines/MAX_EMPLEATS);
   printf("\nNòmina més alta : %.2f € (%s)", vEmpleats[maxNomina].nomina, vEmpleats[maxNomina].nom);
   printf("\nNòmina més baixa : %.2f € (%s)", vEmpleats[minNomina].nomina, vEmpleats[minNomina].nom);
}
```

Com es pot veure, s'utilitza un vector de `tEmpleat` de forma que donada una longitud màxima del vector, anirem introduint els tEmpleats un a un dins d'ell. Una vegada fet, ja podem tornar a recórrer el vector de tEmpleat i realitzar tots els càlculs que ens demanen, així com mostrar per pantalla les nòmines de tots els empleats.

En aquest exemple la variable i fa d'índex per recórrer el vector, i les variables maxNomina i minNomina també són índexos: indiquen en quina posició estàn els empleats amb la nòmina més alta i més baixa respectivament.

6.2 Exemple accions

Adjunto un exemple inventat de com es poden definir accions que permetin modificar els atributs d'una tupla passada com a punter.

```
#include <stdio.h>

#define MAX_CHAR 10+1

typedef struct {
   char nom[MAX_CHAR];
   float nomina;
} tEmpleat;

/* Predeclaració de les accions */

void printEmpleat(tEmpleat empleat);
void setNominaEmpleat(tEmpleat *empleat, float nomina);
void setNomEmpleat(tEmpleat *empleat, char nom[MAX_CHAR]);

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

```
   /* Declarem nouEmpleat de tipus tEmpleat,
      però no li donarem cap valor directament
      als seus dos atributs (nom i nomina): ho
      farem mitjançant dues accions */
   tEmpleat nouEmpleat;

   /* Els valors dels atributs nom i nomina els
      llegirem des de teclat i els desarem inicialment
      en les següents dues variables */
   char nom[MAX_CHAR];
   float nomina;

   printf("\nNom empleat : ");
   scanf("%s", nom);

   printf("Nòmina empleat : ");
   scanf("%f", &nomina);

   /* Assignem el nom i la nòmina al tEmpleat empleat1
      utilitzant les accions definides. Fixeu-vos
      que el paràmetre nouEmpleat el passem com a punter
      (passem la seva adreça en memòria, ja que va
      precedit per &) */
   setNomEmpleat(&nouEmpleat, nom);
   setNominaEmpleat(&nouEmpleat, nomina);

   /* Mostrem les dades de la tupla tEmpleat empleat1
      per pantalla */
   printEmpleat(nouEmpleat);

   return 0;
}

/* Implementació de les accions */

void printEmpleat(tEmpleat empleat) {

   /* El paràmetre empleat és d'entrada, amb el
      que l'accés als seus atributs ho farem
      amb un punt : empleat.nom, empleat.nomina */
   printf("\nDades de l'empleat: \n");
   printf("\tNom: %s\n", empleat.nom);
   printf("\tNòmina: %.2f €\n", empleat.nomina);
}

void setNominaEmpleat(tEmpleat *empleat, float nomina) {

   /* El paràmetre empleat (de tipus inout) és un punter,
      per tal que des de dins de l'acció sigui possible
      modificar el valor (d'un atribut) de l'empleat
      definit al main del nostre programa.
      L'accés a un atribut d'un element referenciat amb
      un punter es fa amb '->' : empleat->nomina. */
```

```
    empleat->nomina = nomina;
}

void setNomEmpleat(tEmpleat *empleat, char nom[MAX_CHAR]) {

   /* Idem que en l'acció setNominaEmpleat. En aquest cas
      a més a més cal recordar que l'assignació d'strings
      la fem amb la funció strcpy de C, en comptes
      d'utilitzar l'assignació habitual dels tipus
      primitius (char, int, float, ...) */
   strcpy(empleat->nom, nom);
}
```

6.3 Comanda scanf

Quan utilitzem la comanda `scanf` fins ara sempre li hem passat el nom de la variable precedit per`&`. Això significa que realment a aquesta comanda li estem passant la posició de la memòria on resideix la variable que li indiquem.

Així, per exemple, quan fem la següent operació `scanf("%d", &numero);` estem passant el valor que introduïm per teclat directament a la posició de memòria on tenim desada la variable numero. D'aquí ve utilitzar `&numero` en comptes de `numero`. El mateix comportament tenim pels tipus primitius `char`, `float`, etc.

Els vectors de caràcters en llenguatge C tenen una característica: el nom de l'array conté l'adreça de memòria on està desada la primera posició de l'array.

Per exemple, quan executem `scanf("%s", cadena);` el valor de cadena és l'adreça de memòria inicial on està ubicat l'array. Dit d'una altra manera, cadena conté el mateix valor que `&cadena[0]` (és una altra forma que tenim per referir-nos a la posició inicial en memòria de l'array).

Adjunto un exemple amb tots aquests conceptes:

```
#include <stdio.h>
#define MAXIM 10

int main(int argc, char **argv) {

    char cadena[MAXIM];
    int numero;

    printf("Reservada la posició de memòria %p per la variable numero\n", &numero);
    printf("Reservada la posició de memòria %p per la variable cadena\n", cadena);
    printf("Reservada la posició de memòria %p per la variable cadena\n", &cadena[0]);

    printf("\nIntrodueix un número enter: ");
    scanf("%d", &numero);
    printf("\nIntrodueix una cadena: ");
    scanf("%s",cadena);

    printf("\nHas assignat els següents valors :\n");
    printf("numero = %d \n", numero);
```

```
    printf("cadena = %s \n", cadena);

    return 0;
}
```

6.4 Finalitzador '\0'

Com es va veure al mòdul **Cadenes de caràcters en C** de la xWiki, *"una cadena de caràcters o string és una seqüència de caràcters finalitzada pel caràcter '\0'"*. Per tant hem de tenir en compte que el finalitzador '\0' càpiga a la nostra variable, ja que aquesta és la forma que té C de saber on s'acaba un string en memòria.

Imaginem que tenim tres cadenes, amb el mateix contingut però de mida diferent (podem tenir posicions buides). Què passa si les comparem? I si comparem amb una cadena de mateix contingut però sense finalitzador '\0'?

```c
#include <stdio.h>
#include <string.h>

int main() {

    char ciutat1[7] = "Girona";
    char ciutat2[8] = "Girona";
    char ciutat3[6] = "Girona"; /* no conté el '\0' final */

    /* si strcmp retorna 0 significa que les dues cadenes són iguals */
    printf ("Les variables ciutat1 i ciutat2 són iguals? %d\n", strcmp(ciutat1, ciutat2));
    printf ("Les variables ciutat1 i ciutat3 són iguals? %d\n", strcmp(ciutat1, ciutat3));
}
```

La forma que tenim per forçar que una cadena no contingui el finalitzador és limitant la seva mida als caràcters que contindrà, sense tenir en compte reservar-ne un pel '\0'. En aquest cas ho fem amb char `ciutat3[6] = "Girona"`.

La sortida generada és la següent:

```
Les variables ciutat1 i ciutat2 són iguals? 0
Les variables ciutat1 i ciutat3 són iguals? -1
```

D'aquí la importància del finalitzador de cadenes de caràcters. Per tant, si per exemple ens diuen que tindrem una variable x de tipus string i de mida màxima 15, realment al nostre programa la definirem amb longitud 15+1, per tal que hi càpiga el finalitzador '\0' en cas que s'ocupin els 15 caràcters anteriors.

Chapter 7

PAC06

7.1 Comanda strcmp

La funció `strcmp` de C fa una comparació caràcter a caràcter de dos cadenes i com a resultat:

- Retorna 0: si les dues cadenes són iguals
- Retorna -1: si la primera cadena < segona cadena
- Retorna 1: si la primera cadena > segona cadena

Com funciona una comparació caràcter a caràcter? Imaginem que tenim els següents dos string: "UOC" i "UAB".

La comparació caràcter a caràcter que realitza la funció strcmp és la següent:

- Caràcters de la posició 0 dels dos string: **U**OC vs **U**AB. Són iguals, amb el que passa a comparar el següent caràcter.
- Caràcters de la posició 1 dels dos string: U**O**C vs U**A**B. Són diferents ('O' > 'A'), finalitza la comparació i la funció `strcmp` retorna el valor 1.

Per si vols fer proves, aquest exemple codificat en C podria ser:

```c
#include <stdio.h>
#include <string.h>

#define MAX_STRING 3+1

int main(int argc, char **argv) {

    char cadena1[MAX_STRING] = "UOC";
    char cadena2[MAX_STRING] = "UAB";
    int resultatComparacio = 0;

    resultatComparacio = strcmp(cadena1, cadena2);

    printf("Comparació strings \"%s\" i \"%s\" = %d\n", cadena1, cadena2, resultatComparacio);

    if (resultatComparacio == 0) {
```

```
        printf("El resultat %d significa que l'string \"%s\" == string \"%s\"\n", resultatComparacio, ca
    } else if (resultatComparacio == -1) {
        printf("El resultat %d significa que l'string \"%s\" < string \"%s\"\n", resultatComparacio, cad
    } else if (resultatComparacio == 1) {
        printf("El resultat %d significa que l'string \"%s\" > string \"%s\"\n", resultatComparacio, cad
    }

    return 0;
}
```

En què es basa strcmp per decidir que 'O' > 'A' ? en el valor ASCII (numèric) que té associat cada caràcter. Per aquest motiu és normal que interpreti diferent una 'A' i una 'a', ja que són caràcters diferents; de fet segons els valors ASCII, tenim que 'A' < 'a'.

Per si no vols consultar la taula ASCII per internet, pots obtenir tu mateix els valors de cada caràcter de la següent forma:

```c
#include <stdio.h>

int main(int argc, char **argv) {

    int i = 0;

    /* Relació de caràcters ASCII (només és un subconjunt!)
       ordenats de més petit a més gran */
    for (i=33; i<=126; i++) {
        printf("%d : %c\n", i, i);
    }
}
```

7.2 Exemple

Imaginem que treballem amb els empleats d'una empresa. Posem per cas que després d'introduir n-empleats al nostre sistema, volem una funció que ens retorni l'empleat que té la nòmina més petita.

Com que esperem un valor de retorn, estem davant d'una funció. A la funció li passarem un vector d'empleats amb tots els empleats que prèviament hem introduit a la nostra empresa.

Sense entrar en la codificació de la funció, el main del nostre programa pot ser similar al següent:

```c
#include <stdio.h>
#include <string.h>

#define MAX_EMPLEATS 2
#define MAX_CARACTERS 20+1

typedef struct {
    char nom[MAX_CARACTERS];
```

```
    char cognom[MAX_CARACTERS];
    float nomina;
} tEmpleat;

int main(int argc, char **argv) {

    tEmpleat vEmpleats[MAX_EMPLEATS];
    int i = 0;

    for (i=0; i<MAX_EMPLEATS; i++) {

        printf("\nNom empleat %d : ", i);
        scanf("%s", vEmpleats[i].nom);

        printf("Cognom empleat %d : ", i);
        scanf("%s", vEmpleats[i].cognom);

        printf("Nòmina empleat %d : ", i);
        scanf("%f", &vEmpleats[i].nomina);

    }

    tEmpleat empleat = cercaEmpleatNominaMinima(vEmpleats);
    printf("\nL'empleat amb la nòmina més baixa és %s, %s (%.2f €)", empleat.cognom, empleat.nom, emplea
    return 0;
}
```

Fins aquest punt no hi ha res nou: utilitzem en aquest cas un vector de `tEmpleat` per tal d'anar introduint els empleats per teclat, i per cada empleat demanem el nom, el cognom i la seva nòmina.

El que cal fer ara és implementar la funció `cercaEmpleatNominaMinima`, que rep com a paràmetre el vector d'empleats de l'empresa.

De moment oblidem-nos que estem davant d'una funció, simplement centrem-nos en quina és l'acció que volem fer. En aquest cas, volem fer un programa que recorri un a un tots els elements d'un vector, i trobi l'empleat que cobra menys.

Una possible codificació seria la següent:

```
int i = 0;
int minNomina = 0;

for (i=0; i<MAX_EMPLEATS; i++) {
    if (vector[i].nomina < vector[minNomina].nomina) {
        minNomina = i;
    }
}
```

Aquest fragment de codi simplement recorre un a un tots els `tEmpleat` d'un vector, comparant la nòmina més baixa trobada fins el moment amb la de l'empleat que està tractant en qüestió, i si aquesta segona és més baixa, ens quedem amb la seva posició dins del vector com a empleat amb la nòmina més baixa (és el mateix plantejament que l'exposat a l'exemple de la setmana passada, per tant fins aquí res nou).

Ara convertim aquest fragment de codi en una funció:

```c
tEmpleat cercaEmpleatNominaMinima(tEmpleat vector[MAX_EMPLEATS]) {

    int i = 0;
    int minNomina = 0;

    for (i=0; i<MAX_EMPLEATS; i++) {
        if (vector[i].nomina < vector[minNomina].nomina) {
            minNomina = i;
        }
    }

    return vector[minNomina];
}
```

Com pots veure l'única diferència és que ara el codi té la capçalera de la funció, la qual ens diu que rep com a paràmetre un element de tipus vector de `tEmpleat`, i que retornarà un element de tipus `tEmpleat`.

D'aquesta forma el programa complet queda de la següent manera:

```c
#include <stdio.h>
#include <string.h>

#define MAX_EMPLEATS 2
#define MAX_CARACTERS 20+1

typedef struct {
    char nom[MAX_CARACTERS];
    char cognom[MAX_CARACTERS];
    float nomina;
} tEmpleat;

/* Predeclaració de funcions/accions */
tEmpleat cercaEmpleatNominaMinima(tEmpleat vector[MAX_EMPLEATS]);

int main(int argc, char **argv) {

    tEmpleat vEmpleats[MAX_EMPLEATS];
    int i = 0;

    for (i=0; i<MAX_EMPLEATS; i++) {

        printf("\nNom empleat %d : ", i);
        scanf("%s", vEmpleats[i].nom);

        printf("Cognom empleat %d : ", i);
        scanf("%s", vEmpleats[i].cognom);
```

```
        printf("Nòmina empleat %d : ", i);
        scanf("%f", &vEmpleats[i].nomina);

    }

    tEmpleat empleat = cercaEmpleatNominaMinima(vEmpleats);
    printf("\nL'empleat amb la nòmina més baixa és %s, %s (%.2f €)", empleat.cognom, empleat.nom, emplea

    return 0;
}

/* Implementació de funcions/accions */
tEmpleat cercaEmpleatNominaMinima(tEmpleat vector[MAX_EMPLEATS]) {

    int i = 0;
    int minNomina = 0;

    for (i=0; i<MAX_EMPLEATS; i++) {
        if (vector[i].nomina < vector[minNomina].nomina) {
            minNomina = i;
        }
    }

    return vector[minNomina];
}
```

Aquest exemple pot semblar senzill perquè el tipus de comparació que estem fent és numèrica: comparem dos camps de tipus float (les nòmines de dos empleats).

Ampliem ara la funcionalitat del nostre programa: volem que pugui fer una cerca per cognom entre els nostres empleats. Per fer aquesta cerca, caldrà comparar una cadena de caràcters amb el cognom de cada empleat.

Per no fer la funció més complexa del necessari, imaginarem que cap cognom es pot repetir i que sempre trobarem un cognom com el que busquem (l'objectiu és veure com funciona strcmp). Així doncs la nostra funció rebrà un vector d'empleats i un cognom a cercar, i retornarà l'empleat amb aquell cognom.

Poso tot el codi complet, comentant la part de l'strcmp detalladament (remarco en blau la comparació) :

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

#define MAX_EMPLEATS 2
#define MAX_CARACTERS 20+1

typedef struct {
    char nom[MAX_CARACTERS];
    char cognom[MAX_CARACTERS];
    float nomina;
} tEmpleat;

/* Predeclaració de funcions/accions */
```

```
tEmpleat cercaEmpleatNominaMinima(tEmpleat vector[MAX_EMPLEATS]);
tEmpleat cercaEmpleatPerCognom(tEmpleat vector[MAX_EMPLEATS], char cognom[MAX_CARACTERS]);

int main(int argc, char **argv) {

    tEmpleat vEmpleats[MAX_EMPLEATS];
    char cognom[MAX_CARACTERS];
    int i = 0;

    for (i=0; i<MAX_EMPLEATS; i++) {

        printf("\nNom empleat %d : ", i);
        scanf("%s", vEmpleats[i].nom);

        printf("Cognom empleat %d : ", i);
        scanf("%s", vEmpleats[i].cognom);

        printf("Nòmina empleat %d : ", i);
        scanf("%f", &vEmpleats[i].nomina);
    }

    tEmpleat empleat = cercaEmpleatNominaMinima(vEmpleats);
    printf("\nL'empleat amb la nòmina més baixa és %s, %s (%.2f €)", empleat.cognom, empleat.nom, emplea

    printf("Cognom de l'empleat a cercar : ");
    scanf("%s", cognom);

    tEmpleat empleatCognom = cercaEmpleatPerCognom(vEmpleats, cognom);
    printf("Dades de l'empleat: %s, %s -> %f", empleatCognom.cognom, empleatCognom.nom, empleatCognom.n

    return 0;
}

/* Implementació de funcions/accions */
tEmpleat cercaEmpleatNominaMinima(tEmpleat vector[MAX_EMPLEATS]) {

    int i = 0;
    int minNomina = 0;

    for (i=0; i<MAX_EMPLEATS; i++) {
        if (vector[i].nomina < vector[minNomina].nomina) {
            minNomina = i;
        }
    }
    return vector[minNomina];
}

tEmpleat cercaEmpleatPerCognom(tEmpleat vector[MAX_EMPLEATS], char cognom[MAX_CARACTERS]) {

    int i = 0;
    tEmpleat empleat;

    for (i=0; i<MAX_EMPLEATS; i++) {
```

```
        /* Per comparar strings utilitzarem la funció
           strcmp de C. Aquesta funció compara dues
           cadenes de caràcters, i retorna un valor
           com a resultat de la comparació:
           - si el valor és 0: les dues cadenes són iguals
           - si el valor és -1: la primera cadena < segona cadena
           - si el valor és 1: la primera cadena > segona cadena
           En el nostre cas ens interessa detectar que
           els cognoms siguin iguals, amb el que
           volem controlar que el valor que retorna
           la funció strcmp sigui 0. */
        if (strcmp(vector[i].cognom, cognom) == 0) {
            return vector[i];
        }
    }
}
```

7.3 Tipos de parámetros de las acciones

A grandes rasgos, el concepto de acción es muy similar al de función pero con dos diferencias:

- No devuelve un valor.
- Los parámetros pueden ser de **entrada**, de **salida**, o de **entrada/salida**.

Para entender bien cómo se implementa una acción, relacionaré ejemplos con los distintos tipos de parámetros.

Entrada: son aquellos parámetros que se pasan a una función y de los cuales solamente usaremos su contenido. Esto significa que trabajaremos con ellos "en modo lectura": obtendremos sus valores, generaremos resultados, pero nunca modificaremos su contenido.

Ejemplo:

```
void suma(int num1, int num2) {
    int resultat = num1 + num2;
    printf("suma2: %d + %d = %d\n", num1, num2, resultat);
}
```

En este caso le podemos pasar los parámetros que queramos a la función que nunca se modificarán:

```
int main(int argc, char **argv) {

    int n1 = 3;
    int n2 = 2;

    printf("Valor de n1 = %d\n", n1);
    printf("Valor de n2 = %d\n", n1);
    printf("Inicio ejecución función\n");
```

```
    suma(n1,n2);
    printf("Fin ejecución función\n");
    printf("Valor de n1 = %d\n", n1);
    printf("Valor de n2 = %d\n", n1);
}
```

El resultado de salida será:

```
Valor de n1 = 3
Valor de n2 = 3
Inicio ejecución función
suma: 3 + 2 = 5
Fin ejecución función
Valor de n1 = 3
Valor de n2 = 3
```

Como ves, ni `n1` ni `n2` ha modificado su valor después de ejecutar la función: son **parámetros de entrada**.

Salida: Son todo lo opuesto a los que acabamos de tratar: a diferencia de los de entrada, los de salida se usan solamente para guardar un valor. Puede contener el valor inicial que sea, que este no se usa para nada dentro de la función a excepción de guardar el resultado final de la operación.

Ejemplo:

```
void suma(int num1, int num2, int *res) {
    *res = num1 + num2;
}
```

De momento ignoremos que se trata de un puntero: independientemente del valor que tenga res al pasarse por parámetro, a su salida contendrá la suma de los otros dos parámetros de entrada `num1` y `num2`.

Si ejecutamos el código:

```
int main(int argc, char **argv) {

    int n1 = 3;
    int n2 = 2;
    int resultat = 0;
    int *pResultat = &resultat;

    printf("Valor de resultat = %d\n", resultat);
    printf("Inicio ejecución función\n");
    suma(n1,n2,pResultat);
    printf("suma: %d + %d = %d\n", n1, n2, resultat);
    printf("Fin ejecución función\n");
    printf("Valor de resultat = %d\n", resultat);
}
```

Obtenemos:

```
Valor de resultat = 0
Inicio ejecución función
suma: 3 + 2 = 5
Fin ejecución función
Valor de resultat = 5
```

El valor de resultat ha cambiado, de 0 a 5. Se considera un parámetro de salida porque, independientemente del valor inicial que tenga, este no se usa para nada y al finalizar la función contendrá el resultado de realizar una operación.

Ahora sí comentamos el uso de un puntero: la forma que tenemos en C para modificar una variable definida fuera de una función desde dentro de la misma es trabajando precisamente con su dirección de memoria.

Entrada/salida: este tipo de parámetro es una suma de los dos anteriores: por un lado su valor importa para realizar una operación, y a su vez al finalizar la ejecución de la función tendrá un valor distinto, también significativo.

```c
void suma(int *pNum1, int num2) {

    *pNum1 = *pNum1 + num2;
}
```

En este caso calculamos la suma como `n1 = n1 + n2` : el resultado de la suma de las dos variables lo guardaremos en la primera de ellas.

Ejecutamos el código:

```c
int main(int argc, char **argv) {
    int n1 = 3;
    int n2 = 2;
    int *pNum1 = &n1;

    printf("Valor de n1 = %d\n", n1);
    printf("Valor de n2 = %d\n", n2);
    printf("Inicio ejecución función\n");
    suma(pNum1,n2);
    printf("Resultado suma = %d\n", n1);
    printf("Fin ejecución función\n");
    printf("Resultado (n1=n1+n2) = %d\n", n1);
}
```

Y ahora la salida será:

```
Valor de n1 = 3
Valor de n2 = 2
Inicio ejecución función
Resultado suma = 5
Fin ejecución función
Resultado (n1=n1+n2) = 5
```

En este caso el valor de la variable `n1` importa tanto en la entrada como en la salida de la función, por lo que se trata de un parámetro de entrada/salida.

7.4 Exemple

Adjunto un exemple inventat de com es poden definir accions que permetin modificar els atributs d'una tupla passada com a punter. He posat comentaris detallats al codi per tal que sigui el més clar possible:

```c
#include <stdio.h>

#define MAX_CHAR 10+1

typedef struct {
    char nom[MAX_CHAR];
    float nomina;
} tEmpleat;

/* Predeclaració de les accions */

void printEmpleat(tEmpleat empleat);
void setNominaEmpleat(tEmpleat *empleat, float nomina);
void setNomEmpleat(tEmpleat *empleat, char nom[MAX_CHAR]);

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

    /* Declarem nouEmpleat de tipus tEmpleat,
       però no li donarem cap valor directament
       als seus dos atributs (nom i nomina): ho
       farem mitjançant dues accions */
    tEmpleat nouEmpleat;

    /* Els valors dels atributs nom i nomina els
       llegirem des de teclat i els desarem inicialment
       en les següents dues variables */
    char nom[MAX_CHAR];
    float nomina;

    printf("\nNom empleat : ");
    scanf("%s", nom);

    printf("Nòmina empleat : ");
    scanf("%f", &nomina);
```

```
    /* Assignem el nom i la nòmina al tEmpleat nouEmpleat
       utilitzant les accions definides. Fixeu-vos
       que el paràmetre nouEmpleat el passem com a punter
       (passem la seva adreça en memòria, ja que va
       precedit per &) */
    setNomEmpleat(&nouEmpleat, nom);
    setNominaEmpleat(&nouEmpleat, nomina);

    /* Mostrem les dades de la tupla tEmpleat nouEmpleat
       per pantalla */
    printEmpleat(nouEmpleat);
    return 0;
}

/* Implementació de les accions */

void printEmpleat(tEmpleat empleat) {

    /* El paràmetre empleat és d'entrada, amb el
       que l'accés als seus atributs ho farem
       amb un punt : empleat.nom, empleat.nomina */
    printf("\nDades de l'empleat: \n");
    printf("\tNom: %s\n", empleat.nom);
    printf("\tNòmina: %.2f €\n", empleat.nomina);
}

void setNominaEmpleat(tEmpleat *empleat, float nomina) {

    /* El paràmetre empleat (de tipus inout) és un punter,
       per tal que des de dins de l'acció sigui possible
       modificar el valor (d'un atribut) de l'empleat
       definit al main del nostre programa.
       L'accés a un atribut d'un element referenciat amb
       un punter es fa amb '->' : empleat->nomina. */
    empleat->nomina = nomina;
}

void setNomEmpleat(tEmpleat *empleat, char nom[MAX_CHAR]) {

    /* Idem que en l'acció setNominaEmpleat. En aquest cas
       a més a més cal recordar que l'assignació d'strings
       la fem amb la funció strcpy de C, en comptes
       d'utilitzar l'assignació habitual dels tipus
       primitius (char, int, float, ...) */

   strcpy(empleat->nom, nom);
}
```

Chapter 8

PAC07

8.1 Estructura

Según lo que se indica en el enunciado de la PAC07, la estructura del proyecto tendría que ser:

```
proyecto
   |
   |-- /include/
   |     |
   |     \-- *.h
   |
   \-- /src/
         |
         |-- main.c
         \-- *.c
```

El detalle correspondiente a este esquema sería:

- Carpeta ./include : contiene el fichero de cabeceras `*.h`. Este fichero tiene la predeclaración de todas las funciones/acciones, así como la definición de todos los enum y struct de nuestro código C, así como las constantes necesarias.
- Carpeta ./src : contiene el código fuente de nuestro proyecto. Dentro del fichero `*.c` tenemos la implementación de las funciones/acciones. Por contra, en el fichero main.c tenemos nuestro programa principal donde vamos pidiendo los datos de los trenes con la acción correspondiente, realizamos el cálculo de longitud de cada tren con la función que hemos implementado, etc.

Así pues el main.c lo necesitas, ya que allí es donde llamas a las acciones y funciones, muestras los resultados por pantalla, etc. Se trata básicamente del núcleo de tu programa, donde ejecutas las acciones/funciones que has predeclarado/implementado en los otros ficheros del proyecto.

8.2 Tipus de paràmetres en accions i funcions

En les **accions** els paràmetres poden ser de tipus **in**, **out** o **inout**. Cal especificar-ho en el moment de definir l'acció al nostre algorisme.

Exemple: tres formes d'implementar una suma de dos enters amb diferents accions segons els tipus **in**, **out** o **inout** dels paràmetres:

```
action suma1(in num1: integer, in num2: integer)
    var
        resultat: integer;
    end var

    resultat := num1 + num2;
    writeString("Resultat de la suma = ");
    writeInteger(resultat);
end action

action suma2(in num1: integer, in num2: integer, out resultat: integer)
    resultat := num1 + num2;
end action

action suma3(inout num1: integer, in num2: integer)
    num1 := num1 + num2;
end action
```

A les funcions en canvi tots els paràmetres són d'entrada, amb el que no cal indicar l'**in**.

Exemple:

```
function suma4(num1: integer, num2: integer): integer
    var
        resultat: integer;
    end var

    resultat := num1 + num2;
    return resultat;
end function
```

8.3 scanf: acció o funció?

El que ens indica la signatura de `scanf` és que estem davant d'una funció:

```c
int scanf(const char *format, type* var1, ...);
```

El segon argument no és de tipus **out**, ja que el que li passem no és la variable que guardarà el valor llegit, sinó el punter a la variable que guardarà el valor llegit, i aquest punter no varia en cap moment.

```c
/* Per exemple, el valor de &numero és 0xbfb86c40 */
scanf("%d", &numero)
/* una vegada executada la funció scanf, el valor de &numero continua sent 0xbfb86c40 */
```

La funció `scanf` com a tal retorna un valor, tot i que nosaltres no l'utilitzem (retorna el número d'elements processats correctament). Dit d'una altra forma: que una funció retorni un valor no ens obliga a recuperar-lo i tractar-lo, tot i que habitualment sí que ho farem.

D'altra banda, el fet de definir un paràmetre d'una funció/acció com a `const` significa que aquest paràmetre dins de l'acció/funció es comportarà com a una constant. Per tant dins de l'àmbit de la funció/acció no es podrà modificar.

Aquest fet ens pot interessar o no. Per exemple, imagina que creem la següent funció per sumar dos enters:

```c
#include <stdio.h>

/* Predeclaració de funcions/acions */
int suma(int numA, int numB);

int main(int argc, char **argv) {
    int a = 10;
    int b = 13;

    int resultat = suma(a, b);
    printf("Resultat: %d + %d = %d\n", a, b, resultat);
    return 0;
}

/* Implementació de funcions/accions */
int suma(int numA, int numB) {
    numA = numA + numB;
    return (numA);
}
```

Dins de l'àmbit de la funció, ens interessa per exemple que el paràmetre `numA` sigui constant? No, ja que si ho definim d'aquesta forma la compilació ens fallarà:

```c
#include <stdio.h>

/* Predeclaració de funcions/acions */
int suma(const int numA, int numB);

int main(int argc, char **argv) {
    int a = 10;
    int b = 13;

    int resultat = suma(a, b);
    printf("Resultat: %d + %d = %d\n", a, b, resultat);
    return 0;
```

```
}

/* Implementació de funcions/accions */
int suma(const int numA, int numB) {
    numA = numA + numB;
    return (numA);
}
```

L'error que obtindrem en intentar compilar el programa serà *"error: assignment of read-only parameter 'numA'"*, ja que dins de la funció suma estem modificant el valor de `numA`.

Molt important: que dins de la funció modifiquem el valor de `numA` no significa que fora de l'àmbit de la funció el valor de la variable a varii (ho podem comprovar al printf que es fa posteriorment).

Per tant l'ús de const en un paràmetre s'hauria de limitar a aquells valors que s'hagin de tractar realment com a constants: per exemple, si passem la constant PI = 3.14159... com a paràmetre, dins de la funció segur que no tenim la necessitat de modificar aquesta constant matemàtica, amb el que en aquest cas és més adient utilitzar const en el paràmetre de la funció.

Com a darrer punt, no s'ha de confondre el fet de definir un paràmetre d'una funció com a `const`, a fer que fora de l'àmbit de la funció el paràmetre corresponent es defineixi com a constant:

```
#include <stdio.h>

/* Predeclaració de funcions/acions */
int suma(int numA, int numB);

int main(int argc, char **argv) {
    const int a = 10;
    int b = 13;

    int resultat = suma(a, b);
    printf("Resultat: %d + %d = %d\n", a, b, resultat);

    return 0;
}

/* Implementació de funcions/accions */
int suma(int numA, int numB) {
    numA = numA + numB;
    return (numA);
}
```

En aquest cas quan compilem no obtindrem cap error, ja que la variable a és una constant fora de la funció `suma`, però el seu valor passat com a paràmetre dins de la funció no és una constant.

8.4 Pas per valor vs pas per referència

El clàssic **pas per valor** correspon als paràmetres de tipus **in**, en els quals passem la variable/valor.

D'altra banda el **pas per referència** consisteix en passar com a paràmetre de tipus **out** o **inout** la direcció de memòria de la variable (punter).

8.5 Exemple: capgirar

Imagina que tenim una tupla `tParaula` la qual té dos camps:

- `cadena` : conté l'string amb el valor de la tParaula
- `numeroCaractersCadena` : conté el número de caracters de la cadena

Implementem l'acció capgirar, la qual fa dues operacions:

- Capgira el camp cadena de la `tParaula`; per exemple, si entrem "Fonaments" el resultat serà "stnemanoF".
- Calcula el valor de `numeroCaractersCadena`; si tenim com a cadena "Fonaments", el valor serà 9.

Volem que per teclat es demani el valor pel camp cadena de dues `tParaula`, i en cadascuna d'aquestes dues `tParaula` volem aplicar l'acció capgirar. Una possible forma d'implementar-ho tot plegat seria la següent:

```c
#include <stdio.h>
#include <string.h>

#define MAX_CHAR 20+1
#define MAX_PARAULES 2

typedef enum {FALSE, TRUE} boolean;

typedef struct {
    char cadena[MAX_CHAR];
    int numeroCaractersCadena;
} tParaula;

/* Predeclaració de l'acció.
   Aquesta acció reb un paràmetre inout de tipus tParaula,
   el qual capgira el camp cadena, i calcula el valor
   corresponent pel camp numeroCaractersCadena */
void capgirar(tParaula *mot);

int main(int argc, char **argv) {
    int i = 0;

    /* Introduim per teclat un total de MAX_PARAULES */
    for (i = 0; i < MAX_PARAULES; i++) {
        tParaula paraula;

        printf("Introdueix una paraula : ");
        scanf("%s", paraula.cadena);

        capgirar(&paraula);
        printf("La paraula capgirada és : %s, de %d lletres.\n", paraula.cadena, paraula.numeroCaracters
    }
}

/* Implementació de l'acció */
void capgirar(tParaula *mot) {
```

```
    int i;
    tParaula motCapgirat;
    int midaMot = strlen(mot->cadena);

    for (i=0; i<midaMot; i++ ) {
        motCapgirat.cadena[(midaMot-1)-i] = mot->cadena[i];
    }

    /* Indiquem el finalitzador de l'string */
    motCapgirat.cadena[midaMot] ='\0';

    strcpy(mot->cadena, motCapgirat.cadena);
    mot->numeroCaractersCadena = midaMot;
}
```

L'acció només rep un paràmetre `tParaula`, no pas dos. Però com que el que volem és executar-lo per cada `tParaula` introduida per teclat, repetim la crida dues vegades dins del bucle (una per cada `tParaula`).

8.6 Exemple: funció isParell

Exemple de funció que retorna un booleà:

```
#include <stdio.h>

typedef enum {FALSE, TRUE} boolean;

/* Predeclaració de la funció isParell, la qual retorna un
   booleà que indica si el número passat per paràmetre és
   parell (TRUE) o no (FALSE). */
boolean isParell(int numero);

int main(int argc, char **argv) {
    int numero;

    printf("Tecleja un número : ");
    scanf("%d", &numero);

    if (!isParell(numero)) {
        printf("El número %d és senar.\n", numero);
    } else {
        printf("El número %d és parell.\n", numero);
    }
}

/* Implementació de la funció */
boolean isParell(int numero) {
    if (numero % 2 == 0) {
        return TRUE;
    } else {
```

```
        return FALSE;
    }
}
```

Chapter 9

PAC08

9.1 Exemple

S'ha d'entendre una taula com un conjunt d'elements dels quals sabem en tot moment quants en tenim.

Imaginem que volem fer un programa que calculi la nota mitjana de les PAC d'una assignatura. Podríem plantejar-ho com a un simple array d'enters, però com que ens agrada poder donar més funcionalitats en un futur al nostre programa, tindrem el següent escenari:

tPac: serà el tipus de dades bàsic que tractarà el nostre programa. Aquesta tupla estarà formada d'una banda per un nom descriptiu de la pac, i d'altra banda per la seva nota numèrica amb decimals. tAssignatura: taula que contindrà elements de tipus tPac. A part d'aques array de tPac, també tindrà un comptador intern d'elements: numPacs.

Sobre aquesta base realitzarem dues accions:

- `afegir_pac()`: es tracta d'una acció que inclou un element de tipus `tPac` dins de la taula `tAssignatura`. És una operació similar a l'acció d'omplir taula dels exemples de la xWiki.
- `calcular_nota()`: és una funció que revisa tots els elements `tPac` de la taula `tAssignatura` i en calcula la seva nota mitjana. Es tracta d'una operació equivalent a la dels recorreguts de taula dels exemples de la xWiki, ja que estem recorrent un a un tots els elements `tPac` per obtenir la seva nota.

Una possible forma d'implementar-ho seria la següent:

```c
#include <stdio.h>
#include <string.h>

/* Definició de constants */
#define MAX_PACS 5
#define MAX_NOM 5+1

/* Definició de la tupla tPac */
typedef struct {
    char nom[MAX_NOM];
    float nota;
} tPac;
```

```c
/* Definició de taula tAssignatura */
typedef struct {
    tPac pac[MAX_PACS];
    int numPacs;
} tAssignatura;

/* Definició funcions/accions */

/* Acció que afegeix un element de tipus tPac a la taula tAssignatura */
void afegir_pac(tAssignatura *assignatura, tPac pac);

/* Funció que calcula la mitjana de totes les tPac que conté la taula
   tAssignatura */
float calcular_nota(tAssignatura assignatura);

/* Programa principal */
int main(int argc, char **argv) {

    /* Definim les variables */
    tAssignatura fp;
    tPac pac1, pac2, pac3;
    float nota;

    /* Inicialitzem les variables */
    nota = 0;
    strcpy(pac1.nom,"PAC01");
    pac1.nota = 10;
    strcpy(pac2.nom,"PAC02");
    pac2.nota = 8.5;
    strcpy(pac3.nom,"PAC03");
    pac3.nota = 7.5;

    /* La inicialització de la taula es fa simplement
       posant a 0 el seu comptador */
    fp.numPacs=0;

    /* Afegim ara les pacs a l'assignatura, que és una taula
       d'elements de tipus tPac */
    afegir_pac(&fp, pac1);
    afegir_pac(&fp, pac2);
    afegir_pac(&fp, pac3);

    /* i ara calculem la nota amb la funció calcular_nota */
    nota = calcular_nota(fp);

    printf("La nota mitjana de les %d PAC és %f\n", fp.numPacs, nota);
    return 0;
}

/* Implementació funcions/accions */

void afegir_pac(tAssignatura *assignatura, tPac pac) {
    /* numPacs conté el número d'elements de tipus tPac
```

```
       que conté la taula en cada moment */
    assignatura->pac[assignatura->numPacs] = pac;

    /* Una vegada hem assignat un element nou tPac a la taula
       incrementem el valor de numPacs */
    assignatura->numPacs = assignatura->numPacs + 1;
}

float calcular_nota(tAssignatura assignatura) {
    /* La variable suma conté el sumatori de totes
       les notes de les tPac que estan dins de la taula
       tAssignatura */
    float suma = 0;

    /* Es recorren tots els elements tPac de tAssignatura
       per tal d'obtenir la seva nota i acumular-les a la
       variable suma */
    for (int i = 0; i < assignatura.numPacs; i++) {
        suma = suma + assignatura.pac[i].nota;
    }

    /* Per calcular la mitjana es divideix el sumatori de
       notes pel total d'elements de la taula tAssignatura */
    return suma/assignatura.numPacs;
}
```

Per facilitar la lectura s'ha unit tot el programa en un únic bloc de codi (un únic arxiu).

9.2 Exemple

He adaptat l'exemple anterior per tal que demani els valors iterativament:

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

/* Definició de constants */
#define MAX_PACS 10
#define MAX_NOM 10+1

/* Definició de la tupla tPac */
typedef struct {
    char nom[MAX_NOM];
    float nota;
} tPac;

/* Definició de taula tAssignatura */
typedef struct {
    tPac pac[MAX_PACS];
    int numPacs;
```

```
} tAssignatura;

/* Definició funcions/accions */

/* Acció que afegeix un element de tipus tPac a la taula tAssignatura */
void afegir_pac(tAssignatura *assignatura, tPac pac);

/* Funció que calcula la mitjana de totes les tPac que conté la taula
   tAssignatura */
float calcular_nota(tAssignatura assignatura);

/* Programa principal */
int main(int argc, char **argv) {

    /* Definim les variables */
    tAssignatura fp;
    float nota;
    int numPacs, i;

    /* Inicialitzem les variables */
    nota = 0;
    numPacs = 0;
    i = 0;

    /* Inicialitzem la taula */
    fp.numPacs=0;

    /* Introduim ara les dades de les PAC des de teclat */
    printf("Número de PACs a introduir (<%d): ", MAX_PACS);
    scanf("%d", &numPacs);

    for (i = 0; i < numPacs; i++) {
        tPac pacAux;

        printf("Dades de la PAC0%d : \n", i+1);
        printf("\tNom : ");
        scanf("%s", pacAux.nom);
        printf("\tNota : ");
        scanf("%f", &pacAux.nota);

        /* Afegim a la taula la tPac auxiliar utilitzada
           dins del bucle. En cada iteració es construirà
           i s'afegirà una tPac diferent */
        afegir_pac(&fp, pacAux);
   }

   /* Calculem la nota amb la funció calcular_nota */
   nota = calcular_nota(fp);

   printf("La nota mitjana de les %d PAC és %f\n", fp.numPacs, nota);
   return 0;
}
```

```
/* Implementació funcions/accions */
void afegir_pac(tAssignatura *assignatura, tPac pac) {

    /* numPacs conté el número d'elements de tipus tPac
       que conté la taula en cada moment */
    assignatura->pac[assignatura->numPacs] = pac;

    /* Una vegada hem assignat un element nou tPac a la taula
       incrementem el valor de numPacs */
    assignatura->numPacs = assignatura->numPacs + 1;
}

float calcular_nota(tAssignatura assignatura) {
    /* La variable suma conté el sumatori de totes
       les notes de les tPac que estan dins de la taula
       tAssignatura */
    float suma = 0;

    /* Es recorren tots els elements tPac de tAssignatura
      per tal d'obtenir la seva nota i acumular-les a la
      variable suma */

    for (int i = 0; i < assignatura.numPacs; i++) {
        suma = suma + assignatura.pac[i].nota;
    }

    /* Per calcular la mitjana es divideix el sumatori de
       notes pel total d'elements de la taula tAssignatura */
   return suma/assignatura.numPacs;
}
```

9.3 Com inicialitzar una taula

Per inicialitzar/esborrar una taula únicament ens cal indicar que el nombre d'elements que conté és 0.

La pregunta que ens podem fer és *"simplement inicialitzant a 0 aquest atribut és suficient?"*. La resposta és afirmativa: l'atribut que conté el nombre d'elements d'una taula sempre és l'utilitzat a l'hora de recórrer una taula, ja que ens indica quin és l'últim element de la taula. De la mateixa manera, quan inserim un element incrementarem en 1 el seu valor.

Què passa quan li donem valor 0? Estem indicant que la taula té 0 elements, amb el que quan n'hi afegim un de nou ho farem a la primera posició, sobreescrivint tot el que prèviament hi pogués haver en memòria.

Chapter 10

PAC09

10.1 Exemple

El plantejament és el següent: imaginem que tenim una llista de cartes de tipus `DIAMANTS`, `CORS`, `TREVOLS` i `PIQUES`; el que volem aconseguir és filtrar aquesta llista per un tipus determinat de carta, `DIAMANTS`, i afegir totes aquestes cartes de `DIAMANTS` a una altra llista.

He detallat al màxim el codi amb comentaris, per tal que quedi el més clar possible:

```c
#include <stdio.h>

/* Definició del model de cartes segons l'enllaç:
   https://ca.wikipedia.org/wiki/Joc_de_cartes#Joc_de_cartes_francès */

#define MAX_CARTES 54+1
#define MAX_DIAMANTS_CARTES 13+1

typedef enum {FALSE, TRUE} boolean;

/* El terme "coll" equival a "baraja" */
typedef enum {DIAMANTS, PIQUES, TREVOLS, CORS} tColl;

typedef struct {
    char valor;
    tColl coll;
} tCarta;

typedef struct {
    tCarta cartes[MAX_CARTES];
    int nCartes;
} tCartesList;

/* Predeclaració de les accions i les funcions */
void createList();
void insert(tCartesList *llista, tCarta carta, int index);
```

```
void delete(tCartesList *llista, int index);
tCarta get(tCartesList llista, int index);
boolean end(tCartesList llista, int pos);
boolean emptyList(tCartesList llista);
boolean fullList(tCartesList llista);
void printList(tCartesList llista);
void getCartesByColl(tCartesList llista, tColl coll, tCartesList *llistaByColl);

/* Programa principal */
int main(int argc, char **argv) {

    tCarta carta1, carta2, carta3, carta4, carta5;
    tCartesList llistaCartes, llistaCartesDiamants;

    /* Creem les dues llistes */
    createList(&llistaCartes);
    createList(&llistaCartesDiamants);

    /* Definim una sèrie de cartes */
    carta1.valor = '3';
    carta1.coll = CORS;
    carta2.valor = 'A';
    carta2.coll = DIAMANTS;
    carta3.valor = 'J';
    carta3.coll = TREVOLS;
    carta4.valor = '5';
    carta4.coll = DIAMANTS;
    carta5.valor = 'Q';
    carta5.coll = PIQUES;

    /* I les afegim a la llista genèrica de cartes */
    insert(&llistaCartes, carta1, 0);
    insert(&llistaCartes, carta2, 1);
    insert(&llistaCartes, carta3, 2);
    insert(&llistaCartes, carta4, 3);
    insert(&llistaCartes, carta5, 4);

    /* Mostrem el contingut de la llista de cartes
       per pantalla, amb l'acció printList */
    printf("Contingut de la llista 'llistaCartes' :\n");
    printList(llistaCartes);

    /* Ara volem filtrar la llista genèrica de cartes amb
       un dels colls possibles. Concretament volem separar
       de la llista genèrica de cartes aquelles que siguin
       del coll DIAMANTS; ho fem mitjançant la crida a
       l'acció getCartesByColl. Per veure el seu funcionament,
       reviseu el comentari fet en la implementació d'aquesta
       acció */
    getCartesByColl(llistaCartes, DIAMANTS, &llistaCartesDiamants);

    /* I mostrem ara el contingut de la llista que conté
       únicament les cartes del coll DIAMANTS */
```

```
    printf("Contingut de la llista 'llistaCartesDiamants' :\n");
    printList(llistaCartesDiamants);
    return 0;
}

/* Implementació dels mètodes de la llista: he fet un copy/paste
   de la codificació C de l'exemple 19_12 de la xWiki, canviant
   el genèric "elem" per "tCarta", i el genèric "list" per "tCartesList",
   ja que aquests seran els elements amb els que treballarem
   en aquest exemple */
void createList(tCartesList *llista) {
    llista->nCartes = 0;
}

void insert(tCartesList *llista, tCarta carta, int index) {

    int i = 0;
    if (llista->nCartes == MAX_CARTES) {
        printf("\n Full list \n");
    } else {
        for (i=llista->nCartes-1; i>=index; i--) {
            llista->cartes[i+1] = llista->cartes[i];
        }
        llista->nCartes++;
        llista->cartes[index]=carta;
    }
}

void delete(tCartesList *llista, int index) {

    int i;
    if (llista->nCartes == 0) {
        printf("\n Empty list\n");
    } else {
        for (i=index; i<llista->nCartes-1; i++) {
            llista->cartes[i] = llista->cartes[i+1];
        }
        llista->nCartes--;
    }
}

tCarta get(tCartesList llista, int index) {

    tCarta carta;

    if (llista.nCartes == 0) {
        printf("\n Empty list \n");
    } else {
        carta=llista.cartes[index];
    }
    return carta;
}
```

```
boolean end(tCartesList llista, int pos) {
    return (pos >= llista.nCartes);
}

boolean emptyList(tCartesList llista) {
    return (llista.nCartes == 0);
}

boolean fullList(tCartesList llista) {
    return (llista.nCartes == MAX_CARTES);
}

/* A continuació s'implementaran dues noves accions que
   no surten ja a l'exemple 19_12. La primera acció,
   printList, imprimeix per pantalla una llista. La segona
   acció, getCartesByColl, permet fer un filtratge de
   cartes sobre una llista. */
void printList(tCartesList llista) {

    int i;
    tCarta cartaAux;

    for(i = 0; i < llista.nCartes; i++) {
        cartaAux = get(llista, i);
        if (cartaAux.coll == DIAMANTS) {
            printf(" [%c] de DIAMANTS\n", cartaAux.valor);
        } else if (cartaAux.coll == PIQUES) {
            printf(" [%c] de PIQUES\n", cartaAux.valor);
        } else if (cartaAux.coll == TREVOLS) {
            printf(" [%c] de TREVOLS\n", cartaAux.valor);
        } else if (cartaAux.coll == CORS) {
            printf(" [%c] de CORS\n", cartaAux.valor);
        }
    }
}

/* La següent acció, getCartesByColl, rep tres paràmetres:
   tCartesList llista (in) :
   llista sobre la qual aplicarem el filtre

   tColl tipus (in) :
   correspon al tipus de coll que utilitzarem per fer
   el filtratge; per exemple, si com a coll indiquem
   DIAMANTS significa que el filtratge el farem
   sobre les cartes de tipus DIAMANTS

   tCartesList llistaByColl (out) :
   llista de sortida en la qual s'hi inclouran aquelles
   cartes de la llista d'entrada que són del coll tipus */
void getCartesByColl(tCartesList llista, tColl tipus, tCartesList *llistaByColl) {

    int i, j;
    tCarta carta;
```

```
    createList(llistaByColl);
    i = 0;
    j = 0;

    /* Amb un bucle i la funció end, controlem que no
       hem arribat al final de la llista */
   while ( end(llista, i) == FALSE ) {

        /* Obtenim la carta de la posició i */
        carta = get(llista, i);

        /* Si la carta és del coll indicat per tipus,
           s'afegeix a la llista de sortida */
        if (carta.coll == tipus) {
            insert(llistaByColl, carta, j);
            j = j + 1;
        }
        i = i + 1;
    }
}
```

Chapter 11

PAC10

Chapter 12

PR1

12.1 Mode menu vs mode test

El workspace de la PR1 té habilitats dos modes de funcionament/execució. Per activar un mode o un altre fem el següent, amb el workspace de la PR1 obert: **CodeLite** –> **Build** –> **Configuration manager...**

Aquí es mostra un desplegable amb dues opcions:

- **Menu**: és el mode estàndard de funcionament del programa, el qual mostra el menú per pantalla amb les accions que permet realitzar.
- **Test**: s'executen una sèrie de tests per validar que les accions que hem codificat al nostre programa funcionin com s'espera que ho facin.

Tant si s'escull l'opció **Menu** com l'opció **Test**, després hem de fer l'habitual **CodeLite** –> **Build** –> **Build and Run Project** per executar el programa en el mode que hem escollit.

Chapter 13

PR2

13.1 Exemple

Adjunto un exemple inventat on, donada una pila de cartes inicial, el que volem és codificar l'acció `separarDiamants` que ens permeti separar de la pila totes aquelles que són `DIAMANTS`, afegint-les a una nova pila de cartes `DIAMANTS`.

Exemple: si inicialment tenim la pila1:

```
[Q] de PIQUES
[5] de DIAMANTS
[J] de TREVOLS
[A] de DIAMANTS
[3] de CORS
```

Volem que d'una banda la pila1 contingui totes les cartes que no són `DIAMANTS`:

```
[3] de CORS
[J] de TREVOLS
[Q] de PIQUES
```

I d'altra banda una nova pila2 amb totes les cartes `DIAMANTS`:

```
[A] de DIAMANTS
[5] de DIAMANTS
```

He afegit comentaris al codi explicant tot el plantejament:

```c
#include <stdio.h>

/* Definició del model de cartes segons l'enllaç:
   https://ca.wikipedia.org/wiki/Joc_de_cartes#Joc_de_cartes_francès */

#define MAX_CARTES 54+1

typedef enum {FALSE, TRUE} boolean;

/* El terme "coll"" equival a "baraja" */
```

```
typedef enum {DIAMANTS, PIQUES, TREVOLS, CORS} tColl;

typedef struct {
    char valor;
    tColl coll;
} tCarta;

typedef struct {
    tCarta A[MAX_CARTES];
    int nelem;
} tStack;

/* Predeclaració de les accions i les funcions */
void createStack(tStack *s);
void push(tStack *s, tCarta e);
void pop(tStack *s);
tCarta top(tStack s);
boolean emptyStack(tStack s);
boolean fullStack(tStack s);
void printStack(tStack s);
void separarDiamants(tStack *pilaCartes, tStack *pilaDiamants);

/* Programa principal */

int main(int argc, char **argv) {
    tCarta carta1, carta2, carta3, carta4, carta5;
    tStack pilaCartes, pilaCartesDiamants;

    /* Creem dues piles (reviseu el comentari inicial del bloc
       d'implementació de les accions/funcions de la pila). */
    createStack(&pilaCartes);
    createStack(&pilaCartesDiamants);

    /* Definim una sèrie de cartes. */
    carta1.valor = '3';
    carta1.coll = CORS;
    carta2.valor = 'A';
    carta2.coll = DIAMANTS;
    carta3.valor = 'J';
    carta3.coll = TREVOLS;
    carta4.valor = '5';
    carta4.coll = DIAMANTS;
    carta5.valor = 'Q';
    carta5.coll = PIQUES;

    /* I les afegim totes a pilaCartes */
    push(&pilaCartes, carta1);
    push(&pilaCartes, carta2);
    push(&pilaCartes, carta3);
    push(&pilaCartes, carta4);
    push(&pilaCartes, carta5);

    /* Mostrem el contingut de pilaCartes per
```

```
       pantalla, amb l'acció addicional printStack. */
    printf("\nContingut de la pila 'pilaCartes' :\n");
    printStack(pilaCartes);

    /* Ara volem separar de pilaCartes totes
       aquelles cartes que són DIAMANTS, les quals
       formaran part d'una nova pila de cartes. */
    printf("\nSeparem les cartes en dues piles!!\n");

    /* Explicació detallada dins de la implementació
       de l'acció. */
    separarDiamants(&pilaCartes, &pilaCartesDiamants);

    printf("\nContingut de la pila 'pilaCartes' sense DIAMANTS :\n");
    printStack(pilaCartes);
    printf("\nContingut de la pila 'pilaCartesDiamants' :\n");
    printStack(pilaCartesDiamants);
    return 0;
}

/* Implementació de les accions/funcions de la pila: he fet un copy/paste
   de la codificació C de l'exemple 19_04 de la xWiki, canviant
   el genèric "elem" per "tCarta".

   !!! Atenció !!!: hi poden haver diferències amb funcions/accions
   que us demanen a la PR2. De cara a la PR2 heu de codificar aquests
   mètodes segons les indicacions de l'enunciat. */

void createStack(tStack *s) {
    s->nelem=0;
}

void push(tStack *s, tCarta e) {
   if (s->nelem == MAX_CARTES) {
        printf("\n Full stack \n");
    } else {
        s->A[s->nelem]=e; /* first position in C is 0 */
        s->nelem++;
    }
}

void pop(tStack *s) {
    if (s->nelem == 0) {
        printf("\n Empty stack\n");
    } else {
        s->nelem--;
    }
}

tCarta top(tStack s) {
    tCarta e;
    if (s.nelem == 0) {
        printf("\n Empty stack \n");
```

```
    } else {
        e = s.A[s.nelem-1];
    }
    return e;
}

boolean emptyStack(tStack s) {
    return (s.nelem == 0);
}

boolean fullStack(tStack s) {
    return (s.nelem == MAX_CARTES);
}

/* Acció addicional que mostra per pantalla tots
   els elements d'una pila. */
void printStack(tStack s) {

    tCarta cartaAux;

    while (s.nelem > 0) {
        cartaAux = s.A[s.nelem-1];
        if (cartaAux.coll == DIAMANTS) {
            printf("  [%c] de DIAMANTS\n", cartaAux.valor);
        } else if (cartaAux.coll == PIQUES) {
            printf("  [%c] de PIQUES\n", cartaAux.valor);
        } else if (cartaAux.coll == TREVOLS) {
            printf("  [%c] de TREVOLS\n", cartaAux.valor);
        } else if (cartaAux.coll == CORS) {
            printf("  [%c] de CORS\n", cartaAux.valor);
        }
        s.nelem--;
    }
}

/* Acció que rep dos paràmetres:
   - pilaCartes (inout)
   - pilaDiamants (out)
   Aquesta acció separa de pilaCartes (inout) aquelles
   cartes amb coll DIAMANTS, i les afegeix a
   pilaDiamants (out). Així una vegada executada l'acció, tindrem:
   - pilaCartes: contindrà totes les cartes que no són DIAMANTS.
   - pilaDiamants: contindrà tots els DIAMANTS. */
void separarDiamants(tStack *pilaCartes, tStack *pilaDiamants) {

    tCarta cartaAux;
    tStack pilaNoDiamants;

    /* Totes les cartes inicialment estan a pilaCartes.
       Es tracta d'anar afegint els DIAMANTS a la pilaDiamants,
       i els que no són DIAMANTS a la pila temporal pilaNoDiamants.
       Posteriorment assignarem pilaNoDiamants a pilaCartes (inout).*/
```

```
    /* Inicialitzem la pila auxiliar. */
    createStack(&pilaNoDiamants);

    /* Hem de tractar tots els elements de la pila. */
    while (!emptyStack(*pilaCartes)) {
        cartaAux = top(*pilaCartes);
        if (cartaAux.coll == DIAMANTS) {
            push(pilaDiamants, cartaAux);
        } else {
            push(&pilaNoDiamants, cartaAux);
        }

        pop(pilaCartes);
    }

    /* Reassignació de la pila auxiliar pilaNoDiamants
       a pilaCartes, que al cap i a la fi és el paràmetre
       de tipus inout de l'acció. */
    *pilaCartes = pilaNoDiamants;
}
```

Chapter 14

VMWare i CodeLite

14.1 Per què una màquina virtual?

La màquina virtual s'utilitza per tenir un entorn homogeni de programació, tant per part dels estudiants com per part dels consultors, de forma que qualsevol enunciat/solució publicat a les aules de teoria funcioni a tots els estudiants, i que tots els programes que realitzeu es comportin igual als entorns que s'utilitzaran per corregir-los.

Fa uns semestres ens vam trobar amb uns pocs casos en els quals un programa que funcionava correctament al PC d'un estudiant, fallava a l'hora de ser corregit. I també alguns enunciats que en determinats sistemes operatius / versions de compiladors C, tampoc funcionaven correctament. Per aquest motiu es va decidir utilitzar una màquina virtual.

Nosaltres no tenim forma de controlar que realment utilitzeu la FP20181 o bé un CodeLite instal · lat directament al vostre PC, però si no és així es pot donar alguna situació poc desitjable com les que he comentat.

14.2 Primers passos amb CodeLite

Per crear un programa inicial executem els següents passos:

1. Dins de CodeLite només es pot tenir un workspace obert a la vegada, amb el que abans de res ens assegurarem que no en tens cap d'obert: ves a **CodeLite** –> **Workspace** –> **Close Workspace**.
2. Crearem un nou workspace: **CodeLite** –> **Workspace** –> **New Workspace...** –> **Workspace type**: **C++** –> **Workspace name**: el que correspongui; **Workspace Path**: /home/uoc/Documents/codelite/workspaces/ (o qualsevol altre) –> fi
3. Dins d'un workspace hi creem projectes. En el nostre cas, cada projecte correspondrà a una PAC. Per afegir un projecte a un workspace: **CodeLite** –> **File** –> **New** –> **New Project** –> de tipus **Console: Simple executable (gcc)** –> **Project name**: el que correspongui, per exemple PAC01 –> **Compiler**: **GCC**; **Debugger**: **GNU gdb debugger** –> fi
4. El projecte que acabem de crear conté un arxiu de mostra, que si executem mostra el missatge "hello world" per pantalla. Aquest programa en C el podem editar i afegir/treure tot allò que volem. És aquí dins on hem de codificar el programa en C de la PAC01 (correspondria a l'exercici2).
5. Una vegada tinguem el programa creat, l'executem de la següent forma: **CodeLite** –> **Build** –> **Build and run project**. També es pot compilar amb la icona de la toolbar de la fletxa blanca avall amb fons verd, i executar amb la icona de les rodes dentades grises.
6. Quan executem l'opció anterior de **Build and run project**, sempre ho fa sobre el projecte que tinguem actiu. Si tenim més d'un projecte creat al nostre workspace, podem saber quin està actiu perquè es

mostra per pantalla en negreta, a la llista de projects que surt a l'esquerra de CodeLite. Per fer que un projecte sigui l'actiu, només cal que fem doble click sobre el nom del projecte.

7. El resultat de l'execució del programa es mostrarà en una pantalla nova tipus terminal. És important anar tancant aquestes finestres una vegada ja hem comprovat el resultat de l'execució.

14.3 Com activar un projecte

Dins de CodeLite, a la part esquerra se't mostren tots els projects que has creat al teu workspace. Si et fixes amb el nom de tots ells, veuràs que un d'ells està remarcat en negreta; per exemple pots tenir:

PAC01 **PAC02** PAC03 PAC04

Això significa que quan vas a **CodeLite** –> **Build** –> **Build and Run Project**, s'executarà l'acció sobre el project **PAC02**, tot i que per pantalla estiguis mostrant el codi d'un altre project.

Si fas doble click amb el ratolí sobre el nom del project **PAC04**, veuràs que ara passaràs a visualitzar:

PAC01 PAC02 PAC03 **PAC04**

A partir d'aquest moment, el **Build and Run Project** s'aplicarà sobre **PAC04**.

14.4 Canviar idioma del teclat

Dins de l'escriptori de Lubuntu, clica amb el botó dret del ratolí sobre la barra grisa superior –> **Add/Remove Panel Items** –> pestanya **Panel Applets** –> botó **Add** –> selecciona el plugin **Keyboard Layout Handler** –> **Add** –> **Close**.

En aquests moments a la part superior dreta se't mostrarà l'idioma definit per defecte pel teclat –> marca sobre la bandera amb el botó dret –> **"Keyboard Layout Handler" settings** –> desmarca l'opció **Keep system layout** –> i aquí ja pots afegir l'idioma que vulguis des del botó **Add**; pots prioritzar un idioma o l'altre posant-lo en primera posició a la llista. Una vegada hagis guardat la configuració desitjada, ja t'haurà canviat la disposició del teclat al nou idioma.

Si has deixat definits varis idiomes a la llista, cada vegada que cliquis sobre la bandera de la part superior dreta, et farà un canvi a l'altre idioma de la llista.

Chapter 15

Altres

15.1 Bilbliografia

Recursos gratuïts:

- *C Notes for Professionals book*: https://books.goalkicker.com/CBook/
- *C Programming*: https://en.wikibooks.org/wiki/C_Programming
- *C Programming Boot Camp*: https://www.gribblelab.org/CBootCamp/
- *The C Book*: https://publications.gbdirect.co.uk//c_book/
- *Programming in C*: http://ee.hawaii.edu/~tep/EE160/Book/PDF/Book.html
- *The ANSI C Programming Language*: https://www.dipmat.univpm.it/~demeio/public/the_c_programming_language_2.pdf