FAQ Laboratori de Fonaments de Programació

Jordi Blasco Planesas

2020-06-08

Contents

Introducció

El present recurs és un recopilatori de qüestions plantejades al **Laboratori de Fonaments de Programació** durant els últims semestres.

Les **FAQ** poden donar resposta a dubtes i errors habituals en el moment d'iniciar-nos en el món de la programació. També es poden utilitzar com a material complementari a la teoria explicada a la **xWiki** de **Fonaments de Programació**.

La lectura de les FAQ no cal realitzar-la seqüencialment: cada apartat respon a una consulta o conjunt de consultes, sense relació necessària entre elles.

El fil conductor són les **PAC** : les diferents respostes donades al **Laboratori de Fonaments de Programació** s'han agrupat en PAC segons el moment en que es van produïr. Això no significa que una resposta d'una PAC anterior o posterior a la que estiguem tractant, no ens pugui ser d'utilitat.

Donada aquesta independència de contingut és recomanable utilitzar el cercador del portal: per exemple, si volem cercar consultes on s'hagi tractat `strlen`, simplement posem aquesta comanda al cercador i obtindrem la relació d'entrades que la contenen.

És possible descarregar una versió en PDF/EPUB de totes les FAQ, des del botó de descàrrega de la part superior.

La documentació utilitza les següents icones per referenciar els blocs:

 Indica que el codi mostrat està en **llenguatge algorísmic** i és **correcte**.

 Indica que el codi mostrat està en **llenguatge algorísmic** i és **incorrecte**, conté algun error.

 Indica que el codi mostrat està en **llenguatge C** i és **correcte**.

 Indica que el codi mostrat està en **llenguatge C** i és **incorrecte**, conté algun error.

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

Chapter 1

PAC01

1.1 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:

```c
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.

1.2 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:

```
algorithm lecturaLlum
    var
        lecturaMensual: integer;
    end var

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

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
#include <stdio.h>

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

```
    printf("Introdueix la lectura mensual de la llum (kWh): ");
    scanf("%d", &lecturaMensual);
    return 0;
}
```

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

1.3 Equivalències llenguatge algorísmic vs llenguatge C

A continuació s'indiquen algunes de les equivalències existents entre llenguatge algorísmic i el llenguatge de programació C:

	Llenguatge algorísmic	Llenguatge C
Segueix unes normes?	sí	sí
Es pot compilar?	no	sí
Es pot executar?	no	sí
Assignació de valors a variables	`:=`	`=`
Tipus booleà	`boolean`	`bool`
Tipus enter	`integer`	`int`
Tipus decimal	`real`	`float`
Tipus caràcter	`char`	`char`
Operador igual	`=`	`==`
Operador diferent		`!=`
Operador major	`>`	`>`
Operador major o igual		`>=`
Operador menor	`<`	`<`
Operador menor o igual		`<=`
Operador lògic de conjunció	`and`	`&&`
Operador lògic de disjunció	`or`	`||`
Operador lògic de negació	`not`	`!`

1.4 Impressió de valors incorrecta

Quan es mostra per pantalla el contingut d'alguna variable amb `printf()` és important eliminar el prefix `&` de la variable. Per exemple, si no ho fem tenim que:

```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
```

Per quin motiu obtenim el valor estrany en l'identificador d'avió? Quan fem referència a `&idAvio` estem obtenint realment la posició de memòria on resideix la variable `idAvio`, no pas el valor de la variable. 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
```

1.5 Com definir un enumeratiu

La definició d'un tipus enumeratiu en llenguatge algorísmic 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.

1.6 Com utilitzar un enumeratiu

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

Per exemple, podem tenir la següent definició:

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

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

Una possible forma d'utilitzar els enumeratius es llegir un enter i comparar-lo amb l'element corresponent definit dins de l'`enum`, per tal de realitzar una acció o una altra. Una possible implementació en llenguatge C seria:

```c
#include <stdio.h>

typedef enum {MALE, FEMALE} tGender;

int main(int argc, char **argv) {
    tGender gender;

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

    if (gender == MALE) {
        printf("Patient gender MALE\n");
    } else {
        if (gender == FEMALE) {
            printf("Patient gender FEMALE\n");
        } else {
            printf("Incorrect option\n");
        }
    }
    return 0;
}
```

1.7 Especificador d'un enumeratiu

Els enumeratius en llenguatge C, `enum` utilitzen l'especificador `%u`.

Exemple:

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

```c
typedef enum {PRIVAT, PUBLIC} tTransport;

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

    printf("Amb quin tipus de transport 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);
    return 0;
}
```

1.8 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 una possible forma per fer-ho és 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:

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

1.9 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:

```
#include <stdio.h>

int main(int argc, char **argv) {}
    /* Variable que contindrà el pes d'una persona */
    float pes;

    /* 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.
```

1.10 Frequently Made Mistakes

1.10.1 Definició de tipus: tipus booleà

En llenguatge algorísmic, el tipus `boolean` és un tipus bàsic, i com a tal no cal definir-lo en un bloc `type`.

Pseudocodi incorrecte:

```
type
    boolean = {FALSE, TRUE};
end type
var
    myNum: integer;
    myBool: boolean;
end var
```

Es poden declarar variables de tipus booleà directament, igual que si fos un enter, un real o un caràcter.

Pseudocodi correcte:

```
var
    myNum: integer;
    myBool: boolean;
end var
```

1.10.2 Estil i format: absència d'estil i format en llenguatge algorísmic

Aquest no és un error sintàctic o semàntic, sinó una mala pràctica de disseny i programació molt freqüent.

Pseudocodi incorrecte:

```
action readHotel(out room: integer, out price: real)
var
i: integer;
```

```
end var
writeString("Enter room:");
readInteger(room);
writeString("Enter price:");
readReal(totalPrice);
if(price>MAXPRICE) then
writeString("Invalid price");
end if
for i=1 to rooms do
writeString("Room: ");
writeInteger(i);
end for
end action
```

En el cas del llenguatge algorísmic, les regles de format i d'estil són arbitràries i fixades per conveni, però cal seguir-les perquè sigui fàcil de llegir i revisar, de la mateixa forma que es fa quan es programa en C o altres llenguatges. En l'exemple anterior, no hi ha aplicada cap indentació i el pseudocodi és molt difícil de llegir. Fixeu-vos com canvia quan apliquem correctament unes mínimes regles.

Pseudocodi correcte:

```
action readHotel(out room: integer, out price: real)
    var
        i: integer;
    end var

    writeString("Enter room:");
    readInteger(room);
    writeString("Enter price:");
    readReal(totalPrice);

    if (price > MAXPRICE) then
        writeString("Invalid price");
    end if

    for i:=1 to rooms do
        writeString("Room: ");
    writeInteger(i);
    end for
end action
```

Cal esmentar que les **indentacions** són especialment importants per la lectura

dels programes, ja que permeten identificar ràpidament els blocs de codi, les funcions i accions, les estructures iteratives i alternatives, i la seva dependència jeràrquica. Per aquest motiu, l'ús d'indentacions en el pseudocodi es absolutament necessari.

1.10.3 Declaració de variables: identificadors no permesos

Pseudocodi incorrecte:

```
var
    1Hotel_ID: integer;
    2Hotel_ID: integer;
end var
```

El nom de les variables poden contenir números sempre i quan no estiguin a la primera posició. Utilitzarem el model camelCase per definir el nom de les variables.

Pseudocodi correcte:

```
var
    hotelId1: integer;
    hotelId2: integer;
end var
```

1.10.4 Declaració de variables: operador de declaració

Pseudocodi incorrecte:

```
var
    id:= integer;
    brand:= string;
    name:= string;
end var
```

El següent error podria semblar un error lleu, però és important respectar el **Nomenclator** i utilitzar els operadors correctament. En llenguatge algorísmic,

l'operador de declaració de tipus és `:` i no `:=`, que és l'operador d'assignació de valor.

Pseudocodi correcte:

```
var
    id: integer;
    brand: string;
    name: string;
end var
```

Chapter 2

PAC02

2.1 Booleans en C

Alguns punts a considerar amb els booleans en C:

- Quan utilitzem el tipus `bool` de C ens cal importar la llibreria `<stdbool.h>`, ja que el tipus `bool` no es va definir a les primeres versions del llenguatge C.
- Els valors que pot prendre una variable booleana en C són `false` i `true`. El llenguatge C tracta internament aquests valors com a enters: `false` correspon a `0` i `true` a `1`.
- Quan vulguem introduïr el valor d'un booleà per teclat o bé mostrar-lo per pantalla, utilitzarem l'enter `0` per referir-nos a `false` i `1` per `true`.
- L'especificador de tipus dels booleans és `%d`.
- Per mostrar el valor d'una variable booleana en C ho podem fer de la següent forma:

```c
bool isVocal;
printf("La lletra %c és una vocal (0=false, 1=true) ? %d\n", lletra, isVocal);
```

- Per llegir un booleà des de teclat, ho faríem de la següent forma:

```c
bool variable;
scanf("%d", &variable);
```

- La lectura per teclat d'un booleà, feta tal i com s'indica al punt anterior,

generarà un *warning* del següent tipus: `warning: format '%d' expects argument of type 'int *', but argument 2 has type '_Bool *' [-Wformat=]`. Aquest avís significa que estem utilitzant un especificador de tipus (`%d`) diferent del que li correspondria al tipus `bool`, el qual treballa únicament amb 1 bit. Com que C no disposa de cap especificador de tipus que treballi només amb 1 bit, podem fer dues coses:

- Utilitzar una variable auxiliar que ens ajudi a fer una conversió intermitja a `int`, per tal de transformar posteriorment el valor a `bool`:

```c
bool variable;
int aux;
scanf("%d", &aux);
variable = aux;
```

- Ignorar el warning d'aquesta situació específica: tot i l'avís, el programa es pot compilar i executar.

2.2 Booleans definits com a enumeratius

En semestres anteriors de l'assignatura de **Fonaments de Programació**, s'utilitzava un **enumeratiu** per definir el tipus booleà:

```c
typedef enum {FALSE, TRUE} boolean;
```

Aquesta forma de definir el tipus booleà és **obsoleta** i **no s'utilitza** aquest semestre; tal i com s'ha comentat a l'apartat Booleans en C, els booleans els definirem mitjançant la llibreria `<stdbool.h>`. Tingueu-ho present quan consulteu PAC[1], PR[2] i PS [3] de semestres anteriors, en els quals s'utilitzava la nomenclatura ara obsoleta.

2.3 Constants: 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:

[1] PAC: Prova d'Avaluació Continua
[2] PR: Pràctica
[3] PS: Prova de Síntesi

- `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");
    }
    return 0;
}
```

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");
    }
    return 0;
```

```
}
```

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");
    }
    return 0;
}
```

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.

2.4 Com mostrar el valor d'una constant

En C podem mostrar per pantalla el valor d'una constant definida amb `#define` mitjançant `printf()`. Exemple:

```c
#include <stdio.h>

#define WORD "world"
#define YEAR 2021
#define EXCLAMATION '!'

int main(int argc, char **argv) {
    printf("hello %s and happy %d %c\n", PALABRA, YEAR, EXCLAMATION);
    return 0;
}
```

El resultat que es mostrarà per pantalla serà:

```
hello world and happy 2021 !
```

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

2.6 Semàntica d'una expressió

Si recordem el que es comenta al punt **3.2. Semàntica d'una expressió** del mòdul de la xWiki **Tipus bàsics de dades**, hem d'aconseguir que les expressions i comparacions realitzades als algorismes siguin **semànticament correctes**.

Amb un exemple es veurà més clar: tenim el següent algorisme que indica si una persona és major d'edat. Fixeu-vos que l'expressió realitza una comparació entre dos enters: la variable `edat` i el número `17`.

```
var
    edat: int;
    isMajorEdat: boolean;
end var

algorithm serMajorEdat
```

```
    writeString("Introdueix edat del conductor :");
    edat := readInteger();

    isMajorEdat := (edat > 17);

    writeString("El conductor és major d'edat? :");
    writeBoolean(isMajorEdat);

end algorithm
```

Aquesta expressió **és semànticament correcta**.

En canvi, imaginem ara que el nostre algorisme accepta decimals per l'`edat`; per exemple, `19.5` indicaria que l'edat és de 19 anys i 6 mesos. Així tenim el següent plantejament, on ara la variable `edat` és de tipus `real`:

```
var
    edat: real;
    isMajorEdat: boolean;
end var

algorithm serMajorEdat

    writeString("Introdueix edat del conductor :");
    edat := readReal();

    isMajorEdat := (edat > 17);

    writeString("El conductor és major d'edat? :");
    writeBoolean(isMajorEdat);

end algorithm
```

L'algorisme ara **no és correcte** ja que conté una expressió **semànticament incorrecta**, en la qual es compara `edat` (real) amb `17` (enter). Per solucionar-ho, podem utilitzar alguna de les funcions de conversió comentades a l'apartat **4. Funcions de conversió de tipus del mateix mòdul**:

```
{ opció 1: fem que els dos valors siguin de tipus enter }
isMajorEdat := (realToInteger(edat) > 17);
```

```
{ opció 2: fem que els dos valors siguin de tipus real }
isMajorEdat := (edat > integerToReal(17));
```

2.7 Exemples d'expressions

2.7.1 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 número ");
    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>
#include <stdbool.h>

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

```
    bool isParell;

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

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

2.7.2 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>
#include <stdbool.h>

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

int main(int argc, char **argv) {
    bool 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) ? %d\n", esCapDeSetmana);
    return 0;
}
```

Varis punts a considerar:

- Recordem que inicialment en C no existia el tipus booleà. Per poder utilitzar `bool`, i els valors `true` i `false` ens cal importar prèviament la llibreria `<stdbool.h>`.
- L'especificador de tipus d'un `bool` és `%d`.
- Quan definim una variable de tipus `enum` utilitzem l'especificador de tipus `%u`. Ho podríem fer com a `%d`, però 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 `bool` en el llenguatge de programació C que ja hem comentat anteriorment, 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`.

2.7.3 Exemple 3: esVocal

Exemple: volem fer un programa que, entrat un caràcter pel canal d'entrada, ens indiqui si es tracta o no d'una vocal.

Una possible solució per l'algorisme és la següent:

```
var
    lletra: char;
    isVocal: boolean;
end var

algorithm esVocal

    writeString("Tecleja una lletra :");
    lletra := readChar();

    { en aquest exemple només tractem les vocals minúscules }
    isVocal := lletra = 'a' or lletra = 'e' or lletra = 'i' or lletra = 'o' or lletra = 'u';

    writeString("La lletra ");
    writeChar(lletra);
    writeString(" és una vocal? ");
    writeBoolean(isVocal);

end algorithm
```

Com es pot veure el plantejament de l'algorisme és:

- Llegim un caràcter des del canal d'entrada.
- Comparem el caràcter amb `a`, `e`, `i`, `o`, `u`.
  - Si coincideix amb alguna d'aquestes vocals, la variable `isVocal := true`.
  - Si no coincideix amb cap de les vocals, la variable `isVocal := false`.
- Es mostra el resultat per pantalla.

Com ho podem traduir a llenguatge C? Una possible opció és:

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

int main(int argc, char **argv) {
    char lletra;
    bool isVocal;

    printf("Introdueix una lletra : ");
    scanf("%c", &lletra);

    /* en aquest exemple només tractem les vocals minúscules */
    isVocal = lletra == 'a' || lletra == 'e' || lletra == 'i' || lletra == 'o' || llet

    printf("La lletra %c és una vocal (0=FALSE, 1=TRUE) ? %d\n", lletra, isVocal);
    return 0;
}
```

2.7.4 Exemple 4: votacions

Imaginem que hem de fer un programa que validi si una persona pot anar a votar o no; la condició que ens diuen que cal complir és que la persona sigui major d'edat i a més estigui al cens electoral de la localitat on està votant.

L'algorisme podria ser el següent:

```
algorithm votacions
    var
        isMajorEdat: boolean;
        isCensat: boolean;
        isVotar: boolean;
    end var

    writeString("Ets major d'edat (0=FALSE, 1=TRUE) ? : ");
    isMajorEdat := readBoolean();
    writeString("Estàs al cens electoral (0=FALSE, 1=TRUE) ? : ");
    isCensat := readBoolean();

    { expressió }
    isVotar := isMajorEdat and isCensat;
```

```
    writeString("Pots anar a votar (0=FALSE, 1=TRUE) :");
    writeBoolean(isVotar);

end algorithm
```

Una possible implementació en C seria:

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

int main(int argc, char **argv) {
    bool isMajorEdat;
    bool isCensat;
    bool isVotar;

    printf("Ets major d'edat (0=FALSE, 1=TRUE) ? : ");
    scanf("%d", &isMajorEdat);

    printf("Estàs al cens electoral (0=FALSE, 1=TRUE) ? : ");
    scanf("%d", &isCensat);

    { expressió }
    isVotar = isMajorEdat && isCensat;

    printf("Pots anar a votar (0=FALSE, 1=TRUE) : %d\n", isVotar);
    return 0;
}
```

L'execució d'aquest exemple seria:

```
Ets major d'edat (0=FALSE, 1=TRUE) ? : 1
Estàs al cens electoral (0=FALSE, 1=TRUE) ? : 0
Pots anar a votar (0=FALSE, 1=TRUE) : 0
```

2.7.5 Exemple 5: 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>
#include <stdbool.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 */

int main(int argc, char **argv) {
    int numMetalIceCubes;
    bool 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) : %d\n", isPossible);
    return 0;
}
```

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

2.7.6 Exemple 6: 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 */

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;
}
```

2.7.7 Exemple 7: 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>
#include <stdbool.h>

typedef enum {MALE, FEMALE} tGender;

int main(int argc, char **argv) {
    bool isPointGuard; /* Point Guard = base */
    bool isCenter;     /* Center = pivot */
    bool 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("%d\n", isDrafted);
```

```
    return 0;
}
```

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

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

2.8 Frequently Made Mistakes

2.8.1 Interfície d'usuari: absència de textos informatius

El següent no és un error sintàctic o semàntic, sinó un error de disseny molt freqüent.

Pseudocodi incorrecte:

```
writeInteger(room);
writeInteger(totalPrice);
```

En l'exemple mostrat, sembla que la intenció és mostrar pel canal estàndard dues variables (`room` i `totalPrice`), i efectivament les sentències per imprimir-les són correctes. El problema és que l'usuari no té cap informació sobre el significat dels valors que se li mostren. Cal sempre complementar les dades amb missatges informatius, indicant també les unitats quan sigui necessari.

Pseudocodi correcte:

```
writeString("Room number: ");
writeInteger(room);
writeString("Total price [€]: ");
writeInteger(totalPrice);
```

Aquest error de disseny és més greu en el cas (també real), de no posar cap text informatiu a l'hora de demanar a l'usuari que introdueixi dades per teclat.

Pseudocodi incorrecte:

```
readInteger(room);
readReal(totalPrice);
```

L'usuari no té cap informació sobre els valors a introduir (tipus de dades, intervals vàlids, etc.).

Pseudocodi correcte:

```
writeString("Enter room number [1-100]: ");
readInteger(room);
writeString("Enter total price [€]: ");
readInteger(totalPrice);
```

2.8.2 Caràcters: ús de cometes simples

Pseudocodi incorrecte:

```
var
    fastpass: char;
    areaMap: char;
    allowsFastPass: boolean;
end var
{...}
fastPass := (allowsFastPass = y or allowsFastPass = Y) and (areaMap = B or areaMap = C);
{...}
```

En l'algoritme anterior, sembla que es vol comparar la variable `allowsFastPass` amb els caràcters `y`, `Y`, i la variable `areaMap` amb els caràcters `B`, `C`. El problema és que falten les cometes simples per indicar que es tracta de caràcters, i per tant el que fa l'expressió es comparar amb les variables `y`, `Y`, `B` i `C` respectivament (que a més a més, en aquest algoritme no existeixen. Aquest és un error semàntic greu, que pot no provocar cap error de compilació en llenguatge C, i en canvi comportar comportaments inesperats.

Pseudocodi correcte:

```
var
    fastpass: char;
```

```
    areaMap: char;
    allowsFastPass: boolean;
end var
{...}
fastPass := (allowsFastPass = 'y' or allowsFastPass = 'Y') and (areaMap = 'B' or areaMa
{...}
```

2.8.3 Sintaxi pròpia de C: funcions de lectura/escriptura

Pseudocodi incorrecte:

```
writeString("Code value: %d\n", codeValue);
```

S'ha intentat aplicar a la funció `writeString()` en llenguatge algorísmic la sintaxi pròpia de la funció `printf()` de C, la qual cosa evidentment no és correcta. Recordeu que el llenguatge algorísmic és un llenguatge de disseny de propòsit general, que ha de permetre posteriorment codificar en qualsevol llenguatge de programació

Pseudocodi correcte:

```
writeString("Code value");
writeInteger(codeValue);
```

2.8.4 Conversió de tipus: funcions inexistents

Pseudocodi incorrecte:

```
b1:= characterToCode(myChar);
```

Les funcions de conversió de tipus estan definides al **Nomenclator** de l'assignatura, i serveixen per assegurar la coherència semàntica entre els tipus de variables d'una expressió. En aquest cas, la funció `characterToCode()` no existeix, ja que la que hem declarat és `characterToInteger()`. És habitual trobar errors d'aquest tipus, amb diverses variants de noms de funció. Cal evitar-ho i usar el **Nomenclator** com a guia.

Pseudocodi correcte:

```
b1:= characterToInteger(myChar);
```

2.8.5 Expressions: variables intermitges

Aquest no és un error sintàctic o semàntic, sinó una mala pràctica de disseny.

Pseudocodi incorrecte:

```
var
    bool1: boolean;
    bool2: boolean;
    bool3: boolean;
    bool4: boolean;
end var
bool1:= hasGym or hasPool;
bool2:= closToSubway or distanceFromCityCentre < 5;
bool3:= priceDouble <= 100;
boolFin:= bool1 and bool2 and bool3;
```

En el codi anterior, el que volem obtenir al final és una variable de tipus booleà que ens indiqui si un objecte (suposem que un hotel), té piscina o gimnàs, i a més està prop del metro o a menys de 5 km del centre de la ciutat, i a més el preu és inferior a 100 (euros). Volem reunir tota aquesta informació en una variable booleana, perquè segurament això ens indica alguna qualitat de l'objecte (per exemple, que és un bon hotel). Ara bé, fixeu-vos com per arribar al resultat final, s'han declarat fins a tres variables auxiliars, que no tenen cap altra finalitat que construir l'expressió final.

Declarar variables auxiliars o temporals no és una mala pràctica sempre que es faci amb mesura, i aplicant el sentit comú (cosa que només s'adquireix amb la pràctica del disseny i la programació). Recordem que les variables ocupen espai de memòria, i per tant cal ser curosos a l'hora de declarar-les. Una alternativa seria la que presentem a continuació.

Pseudocodi correcte:

```
var
    bool4: boolean;
```

```
end var
boolFin:= hasGym or hasPool and
          closToSubway or distanceFromCityCentre < 5 and
          priceDouble <= 100;
```

Recordem que el delimitador de les sentències és el punt i coma ;, i que hi ha flexibilitat a l'hora d'escriure les sentències en diverses línies (també en C). Finalment, recordar un cop més que no hi ha cap norma que ens indiqui quantes variables auxiliars hem d'utilitzar. En la solució alternativa hem optat per no utilitzar-ne cap i donar un format entenedor a l'expressió, però això caldrà decidir-ho en cada cas, en base a la pràctica i l'experiència.

2.8.6 Declaració de variables: declaració mal ubicada (bloc central de codi)

Pseudocodi incorrecte:

```
for i:= 1 to selectedHotels->nHotels do
    if selectedHotels[i].city = city then
        var
            scorePoints: integer;
        end var
        scorePoints := scorePoints + 1;
    end if
end for
```

Les variables s'han de declarar **al principi** del **bloc de pseudocodi**, ja sigui una funció, una acció o bé un algorisme directament. No és correcte obrir un bloc de declaració de variables dins un bloc central de pseudocodi (i encara menys obrir diversos blocs de declaració a mesura que necessitem variables). El mateix passa en C: les variables s'han de declarar sempre al bloc inicial de codi (ja sigui una funció, acció o algorisme).

Pseudocodi correcte:

```
var
    scorePoints: integer;
end var

for i:= 1 to selectedHotels->nHotels do
```

```
    if selectedHotels[i].city = city then
       scorePoints := scorePoints + 1;
    end if
end for
```

2.8.7 Declaració de variables: declaració mal ubicada (variables globals)

Aquest no és un error sintàctic o semàntic, sinó una mala pràctica de porgramació que cal evitar.

Codi incorrecte:

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

float maxPrice;
int bestHotel;

int main(int argc, char **argv) {
    /* ... */
    return 0;
}
```

Les variables s'han de declarar sempre dins una funció o acció (ja sigui el main o qualsevola altra), i **al principi** del seu **bloc de codi**. No és correcte declarar variables fora de qualsevol funció, ja que es converteixen en **variables globals**, i el seu ús no es considera una bona pràctica de programació. El motiu és que les variables globals dificulten la lectura i la comprensió del codi, i poden provocar errors d'execució inesperats o actualitzacions involuntàries, en no estar protegides dins de funcions o accions.

Codi correcte:

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

int main(int argc, char **argv) {
    float maxPrice;
    int bestHotel;
```

```
    /* ... */
    return 0;
}
```

2.8.8 Tipus booleà: valors numèrics

Codi incorrecte:

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

int main(int argc, char **argv) {
    bool myBool1;
    bool myBool2;
    bool resultBool;
    /* ... */
    resultBool = (myBool == 1) && (myBool2 == 0);
    return 0;
}
```

Si disposem de la llibreria `<stdbool.h>` pel tractament de booleans en C, no té cap sentit que utilizem els seus equivalents numèrics al codi, especialment en el cas de les expressions. En el seu lloc heu d'utilizar les paraules reservades `true` i `false`.

Com a regla general, hem de procurar usar el mínim nombre possible de valors numèrics, i fer servir en el seu lloc variables i constants.

Codi correcte:

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

int main(int argc, char **argv) {
    bool myBool1;
    bool myBool2;
    bool resultBool;
    /* ... */
    resultBool = (myBool == true) && (myBool2 == false);
    return 0;
}
```

2.8.9 Tipus booleà: cadenes de caràcters

Codi incorrecte:

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

int main(int argc, char **argv) {
    bool myBool1;
    bool myBool2;
    bool resultBool;
    resultBool = (myBool =="true") && (myBool2 == "false");
    return 0;
}
```

Si disposem de la llibreria `<stdbool.h>` pel tractament de booleans en C, podem utilitzar les paraules reservades `true` i `false`, sense cap tipus de modificador ni caràcter. Si afegim les cometes, "true" i "false" es converteixen en cadenes de caràcters, que són una cosa molt diferent i no tenen res a veure amb els booleans (ni amb els seus possibles valors). Aquest error molt greu també es dóna en llenguatge algorísmic.

Cal afegir a més, que en C qualsevol valor que no sigui 0 és interpretat com a `true`, per la qual cosa es produiran errors d'execució, ja que les cadenes de caràcters seran interpretades sempre com un valor cert.

Codi correcte:

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

int main(int argc, char **argv) {
    bool myBool1;
    bool myBool2;
    bool resultBool;
    resultBool = (myBool == true) && (myBool2 == false);
    return 0;
}
```

2.8.10 Sintaxi pròpia de C: operadors

Pseudocodi incorrecte:

```
if (acceptable == 1) or (acceptable == 2) then
    writeString("Hotels cannot be compared");
end if
```

L'operador `==` és propi del llenguatge C, i el seu ús llenguatge algorísmic no és correcte. L'operador correcte en aquest cas és `=`.

Pseudocodi correcte:

```
if (acceptable) = 1 or (acceptable = 2) then
    writeString("Hotels cannot be compared");
end if
```

Chapter 3

PAC03

3.1 Com declarar un vector en llenguatge algorísmic

Quan es declara un vector en llenguatge algorísmic no cal declarar una variable per cadascuna de les posicions d'aquest vector.

Per exemple, si volem un algorisme que sumi tres enters continguts dins d'un vector podem fer el següent:

```
const
    NUM_ENTERS: integer = 3;
end const

algorithm sumaEnters

    var
        vectorEnters: vector[NUM_ENTERS] of integer;
        sumaEnters: integer;
    end var

    vectorEnters[1] := 13;
    vectorEnters[2] := 24;
    vectorEnters[3] := 2;

    { Com es pot veure, accedim directament a les posicions del vector }
```

```
    { en comptes de definir una variable per cada posició. }

    sumaEnters := vectorEnters[1] + vectorEnters[2] + vectorEnters[3];

    { Cal recordar també que en llenguatge algorísmic, en un vector de mida n }
    { la primera posició del vector és la 1 i la darrera la n; en canvi en llenguatge C
    { la primera sempre és la 0 i l'última la n-1. }

    writeString("La suma dels 3 enters del vector és : ");
    writeInteger(sumaEnters);

end algorithm
```

3.2 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 tenim el següent programa compilat en C i li passem 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)` podem accedir des de dins del programa a tots els arguments passats; així haurem de:

```
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.

3.3 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**.
- En **llenguatge C**, van des **de la 0 fins a la N-1**.

En tots dos casos **N** fa referència al número total d'elements del vector.

3.4 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: recordem que en C la posició inicial d'un vector sempre és la 0, i la final sempre és N-1. Exemple, per un vector de 3 posicions tindrem:

```c
int vector1[3];
vector1[1] = 13;  /* Posició del vector1 vàlida */
vector1[2] = 24;  /* Posició del vector1 vàlida */
vector1[3] = 48;  /* Posició del vector1 no vàlida! */
/* En canvi la posició 0 del vector,
 * que tenim disponible no l'hem utilitzat!
 */
```

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

```c
int vector2[2];
```

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

```c
vector2[0] = 10;  /* Posició del vector2 vàlida */
vector2[1] = 13;  /* Posició del vector2 vàlida */
vector2[2] = 24;  /* Posició del vector2 no vàlida! */
```

Per tant qualsevol operació amb `vector2[2]` ens generarà l'error indicat. Si volem que el vector contingui 3 elements només cal definir correctament la seva mida:

```c
int vector2[3];
```

3.5 Concatenació en llenguatge algorísmic

A diferència del llenguatge C, en notació algorísmica no està contemplat l'ús d'especificadors que permetin fer concatenacions entre cadenes de caràcters, enters, decimals, etc.

Per tant en llenguatge algorísmic cal trencar els strings amb fragments més petits i que facin referència únicament a un tipus de dades. Per exemple, si es vol mostrar per pantalla el missatge *"L'empleat que cobra més és Marta, i la seva nòmina és 4675.30 €."*, ho farem de la següent forma:

```
writeString("L'empleat que cobra més és ");
writeString(nomEmpleat);
writeString(", i la seva nòmina és ");
writeReal(nomina);
writeString(" €.");
```

En llenguatge C equivaldria a:

```
printf("L'empleat que cobra més és %s, i la seva nòmina és %.2f €.", nomEmpleat, nomina
```

3.6 Importància dels tipus utilitzats en llenguatge C

El resultat de les operacions en llenguatge C depèn del tipus de variable definit i dels tipus de valors utilitzats. A continuació s'exposen tres casos que, segons el tipus que s'hagi definit en les variables utilitzades, donarà un resultat o un altre:

Cas 1:

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int a;
    int b;
    int c;
    int m;

    scanf("%d", &a);
    scanf("%d", &b);
    scanf("%d", &c);

    m=(a+b+c)/3;

    printf("%d", m);
```

```
    return 0;
}
```

En aquest cas si donem els valors `a = 1`, `b = 3`, `c = 4`, el resultat és `m = 2`. El resultat de la divisió serà un enter, ja que tant numerador com denominador estan formats per enters. El resultat enter es desa en una variable entera, i per pantalla obtindrem: `2`.

Cas 2:

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int a;
    int b;
    int c;
    float m;

    scanf("%d", &a);
    scanf("%d", &b);
    scanf("%d", &c);

    m=(a+b+c)/3;

    printf("%f", m);
    return 0;
}
```

Igual que en el cas anterior, el numerador i el denominador de la divisió estan formats per enters, amb el que el resultat serà un enter. En aquest cas el resultat enter el desem en una variable de tipus `float`, amb el que C mostrarà el resultat amb decimals: `2.000000`

Cas 3:

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int a;
    int b;
    int c;
```

```
    float m;

    scanf("%d", &a);
    scanf("%d", &b);
    scanf("%d", &c);

    m=(a+b+c)/3.0;

    printf("%f", m);
    return 0;
}
```

En aquest cas el resultat de la divisió serà un decimal, ja que el denominador conté un decimal (en aquest cas `3.0`). El resultat amb decimals es guarda en una variable de tipus `float`, i per pantalla es mostrarà : `2.666667`.

3.7 Exemple: notaFinal

En aquesta PAC03 es comencen a tractar dos nous aspectes: els **condicionals** i els **vectors**, dins dels quals hi contemplem també els strings.

El següent exemple contempla bastants punts dels que es comenten als mòduls de teoria per la PAC03, amb el que segurament aquest exemple serà força més dens que no la pròpia PAC03. Per tant, si algun aspecte costa d'entendre inicialment no us preocupeu, és normal.

He afegit comentaris detallats dins del propi exemple, per tal que sigui el més entenedor possible. El següent exemple calcula la nota final d'una assignatura en funció d'una sèrie de condicionals i d'operacions:

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

/* Exemple:
 *
 * Volem un programa que calculi la nota final
 * d'una assignatura. La nota final es calcula a partir
 * de l'AC (Avaluació continua) i la nota de la Pràctica:
 *
 * Nota final = 30% AC + 70% Pràctica
 *
 * Una vegada entrades totes les notes, si se'n
```

```
 * detecta alguna que sigui incorrecta (fora del
 * rang [0.0 a 10.0]), es mostrarà un missatge
 * informatiu per pantalla i no es realitzarà cap
 * més operació.
 *
 * L'AC està formada per 3 PAC: PAC1, PAC2, PAC3.
 * La nota de l'AC es calcula mitjançant la
 * mitjana de les 3 PAC.
 *
 * Si la nota de l'AC és inferior a 4, no cal
 * realitzar cap càlcul: l'assignatura queda suspesa.
 *
 * Si la nota de l'AC és superior o igual a 4, es
 * calcula la nota final juntament amb la nota de
 * la Pràctica.
 *
 * La nota final es mostrarà en format "grade letters",
 * segons la següent relació:
 *
 * MH: 10
 * A: de 9.0 a 9.9
 * B: de 7.0 a 8.9
 * C+: de 5.0 a 6.9
 * C-: de 3.0 a 4.9
 * D: de 0.0 a 2.9
 *
 * Els punts que tracta aquest exemple:
 * - definició de vectors
 * - utilització del condicional if-else
 * - condicionals if-else aniuats
 * - assignació d'un string a una variable amb strcpy
 * - reserva espai pel finalitzador '\0'
 */

#define PAC1 0
#define PAC2 1
#define PAC3 2
#define PRA 3
#define MAX_ACTIVITATS 4
#define MAX_CHARS 2+1    /* el +1 correspon al finalitzador '\0' */
#define PES_AC 0.3       /* AC 30% pes de la nota final */
#define PES_PRA 0.7      /* PRA 70% pes de la nota final */

int main(int argc, char **argv) {
    /* Vector que conté les notes de
```

```
     * totes les activitats del curs
     */
    float notes[MAX_ACTIVITATS];

    /* Nota d'avaluació continua: equival
     * a la mitjana de les 3 PAC
     */
    float notaAC;
    float notaFinalNumerica;

    /* String que conté la nota final
     * de l'assignatura (MH, A, B...)
     */
    char notaFinal[MAX_CHARS];

    /* Es demana des de teclat les notes
     * de les 3 PAC i de la PRA
     */
    printf("Nota PAC1 : ");

    /* L'assignació d'un valor es pot
     * fer directament sobre una posició
     * del vector
     */
    scanf("%f", &notes[PAC1]);

    /* Idem per la resta d'activiats */
    printf("Nota PAC2 : ");
    scanf("%f", &notes[PAC2]);
    printf("Nota PAC3 : ");
    scanf("%f", &notes[PAC3]);
    printf("Nota PRA : ");
    scanf("%f", &notes[PRA]);

    /* Primer de tot, comprovem que
     * totes les notes del vector estiguin
     * dins del rang [0.0 .. 10.0]
     */
    if (notes[PAC1] > 10.0 || notes[PAC2] > 10.0 ||
        notes[PAC3] > 10.0 || notes[PRA] > 10.0 ||
        notes[PAC1] < 0.0 || notes[PAC2] < 0.0 ||
        notes[PAC3] < 0.0 || notes[PRA] < 0.0) {

        printf("\n>> Error detectat en una o més notes:");
        printf("\n>> S'atura el càlcul de la nota final.\n");
```

```
    } else {

        /* En aquest punt sabem que totes les notes
         * estan dins del rang [0.0 .. 10.0]
         */

        /* Comprovem ara que la mitjana de les 3 PAC
         * no sigui inferior a 4
         */
        notaAC = (notes[PAC1] + notes[PAC2] + notes[PAC3]) / 3;

        if (notaAC < 4) {

           /* Per donar millor visibilitat, mostrem només
            * el primer decimal de les notes numèriques
            */
           printf("\n>> Nota mínima d'AC insuficient: %.1f", notaAC);
           printf("\n>> S'atura el càlcul de la nota final.\n");

        } else {

            /* En aquest punt totes les notes són correctes,
             * per tant es pot començar amb el càlcul de la
             * nota final
             */
            notaFinalNumerica = notaAC * PES_AC + notes[PRA] * PES_PRA;

            /* Ara falta saber quina "grade letter" correspon
             * a la notaFinalNumerica calculada; ho solucionem
             * amb nous if-else aniuats
             */
            if (notaFinalNumerica <= 2.9) {

                /* Per assignar un string a una variable
                 * de tipus string, utilitzem la comanda
                 * strcpy, no pas '='
                 */
                strcpy(notaFinal, "D");

            } else {
                if (notaFinalNumerica <= 4.9) {
                    strcpy(notaFinal, "C-");

                } else {
                    if (notaFinalNumerica <= 6.9) {
```

```
                        strcpy(notaFinal, "C+");

                    } else {
                        if (notaFinalNumerica <= 8.9) {
                            strcpy(notaFinal, "B");

                        } else {
                            if (notaFinalNumerica <= 9.9) {
                                strcpy(notaFinal, "A");

                            } else {
                                strcpy(notaFinal, "MH");
                            }
                        }
                    }
                }
            }

            /* Per finalitar, mostrem tots els resultats
             * calculats per pantalla
             */
            printf("\n>> Nota AC: %.1f", notaAC);
            printf("\n>> Nota PRA: %.1f", notes[PRA]);
            printf("\n>> Nota final: %s (%.1f)\n", notaFinal, notaFinalNumerica);
        }
    }

    return 0;
}
```

3.8 Frequently Made Mistakes

3.8.1 Strings: declaració com a vectors de caràcters en llenguatge algorísmic

Pseudocodi incorrecte:

```
var
    name: vector[MAX_CHAR] of char;
end var
```

En l'algorisme anterior, s'intenta emular el llenguatge C a l'hora de declarar una variable de tipus `string`, declarant-la com un vector de caràcters. Això és incorrecte i absolutament innecessari, perquè en llenguatge algorísmic **sí** que existeix el tipus `string`, i per tant la seva declaració és molt més senzilla.

Pseudocodi correcte:

```
var
    name: string;
end var
```

3.8.2 Sintaxi pròpia de C: funcions complexes

Pseudocodi incorrecte:

```
function hotelCmp(hotel1: tHotel, hotel2: tHotel): boolean;
    if strcmp(hotel1.brand, hotel2.brand) = 0 then
        {...}
    end if
end function
```

La intenció de l'algorisme és comparar dos camps de tipus string de les variables `hotel1` i `hotel2`. No obstant es fa ús de la funció `strcmp()`, la qual és pròpia de C i no existeix en llenguatge algorísmic.

En llenguatge algorísmic, la comparació de dues cadenes de caràcters és molt més senzilla: es fa directament amb `=`.

Pseudocodi correcte:

```
function hotelCmp(hotel1: tHotel, hotel2: tHotel): boolean;
    if (hotel1.brand = hotel2.brand) then
        { ... }
    end if
```

3.8.3 Estructura alternativa: if's consecutius

Aquest no és un error sintàctic o semàntic, sinó una mala pràctica de disseny.

Pseudocodi incorrecte:

```
if discountHotel >= 0 and discountHotel <= 10 then
    writeString("Invalid data");
end if

if discountHotel > 10 and discountHotel <= 20 then
    writeString("Not bad");
end if

if discountHotel > 20 and discountHotel <= 50 then
    writeString("Good!");
end if
```

L'algorisme anterior vol mostrar un missatge en pantalla en funció del valor de la variable `discountData`. Per aquest propòsit, res millor que una estructura alternativa, però no de la manera com està dissenyada.

Fixeu-vos que s'han construit tres blocs `if`…`end if` independents i consecutius. Això vol dir que durant l'execució, tots els blocs s'avaluaran de forma consecutiva per decidir si cal executar el codi interior o no. Això no és necessari ni desitjable, ja que augmenta el temps d'execució.

Sempre que puguem, cal construir estructures alternatives niades i excloents, de forma que només s'avaluin les condicions estrictament imprescindibles.

Pseudocodi correcte:

```
if discountHotel >= 0 and discountHotel <= 10 then
    writeString("Invalid data");
    else if discountHotel > 10 and discountHotel <= 20 then
        writeString("Not bad");
        else if discountHotel > 20 and discountHotel <= 50 then
            writeString("Good!");
        end if
    end if
end if
```

3.8.4 Estructura alternativa: if's buits

Aquest no és un error sintàctic o semàntic, sinó una mala pràctica de disseny.

Pseudocodi incorrecte:

```
if discountHotel > = 0 then
else
    writeString("Invalid data");
end if
```

L'algorisme anterior vol mostrar un missatge d'error en pantalla si el valor de la variable discountHotel és negatiu, però l'estructura per fer-ho no és gens apropiada. En cas que es compleixi la condició `discountHotel >= 0`, l'estructura alternativa no executa res. Recordem que és possible una estructura `if` sense `else`, i de fet sembla que en aquest cas s'hagi afegit únicament perquè es creia el contrari.

Pseudocodi correcte:

```
if discountHotel < 0 then
    writeString("Invalid data");
end if
```

3.8.5 Estructura alternativa: interrompre un algorisme (I)

Pseudocodi incorrecte:

```
algorithm nameAlgorithm
    if discountHotel < 0 then
        writeString("Invalid data");
        end algorithm;
    else
        writeString("Continue...");
        { ... }
    end if
```

Sovint ens demanen d'interrompre l'execució d'un algorisme en cas que es doni alguna situació, per exemple, un error en l'entrada de dades. A nivell de disseny algorísmic, no hi ha cap funció ni sentència pensada per efectuar aquesta acció de forma explícita. En l'agorisme de l'exemple, s'intenta fer-ho utiltzant la sentència `end algorithm`, però això no és correcte.

Senzillament cal muntar l'estructura alternativa de forma que quan es produeixi l'error, no s'executi cap més sentència, per exemple posant **tot** el codi a executar dins el bloc de l'`else`. Ens hem d'assegurar, això sí, que un cop sortim del bloc `if…else`, l'algorisme no té res pendent per executar.

Pseudocodi correcte:

```
algorithm nameAlgorithm
    if discountHotel < 0 then
        writeString("Invalid data");
    else
        writeString("Continue...");
        { ... }
    end if
end algorithm;
```

3.8.6 Estructura alternativa: interrompre un algorisme (II)

Pseudocodi incorrecte:

```
algorithm nameAlgorithm
    if discountHotel < 0 then
        writeString("Invalid data");
        exit();
    else
        writeString("Continue...");
        { ... }
    end if
end algorithm
```

Hi ha vegades que també es vol emular incorrectament en llenguatge algorísmic algunes funcions de C que interrompen l'execució del codi, com per exemple `exit()`. Això és incorrecte ja que aquesta funció no existeix en llenguatge algorísmic; de fet, en llenguatge C, encara que existeixi, també cal evitar-la, ja que el seu ús no és una bona pràctica de programació.

Pseudocodi correcte:

```
algorithm nameAlgorithm
    if discountHotel < 0 then
        writeString("Invalid data");
    else
        writeString("Continue...");
        { ... }
    end if
end algorithm;
```

3.8.7 Constants i nombres: Valors numèrics al codi (hard-code)

Pseudocodi incorrecte:

```
const
    NUM_SEATS1: integer = 34;
    NUM_RIDES: integer = 3;
    A1: integer = 1;
    A2: integer = 2;
    A3: integer = 3;
end const

var
    emptySeats[3]: vector of integer;
end var

{ input values }
writeString("EMPTY SEATS  ");
writeString(NAME_RIDE1);
writeString("  >> ");
emptySeats[1] := readInteger();
```

En l'algoritme anterior, es declara el vector d'enters `emptySeats` amb una longitud igual a 3 posicions. Posteriorment, es llegeix un enter i es guarda a la primera posició del vector. A l'enunciat de l'exercici es donaven unes constants ja declarades, i es demanava explícitament utilitzar-les per operar amb els vectors.

El motiu no és altre que introduir el (bon) costum de fer servir constants i evitar

al màxim els valors numèrics directes al codi (tècnica coneguda com *hardcode*). D'aquesta manera, és molt més fàcil mantenir el codi posteriorment i fer-hi modificacions.

En cas que volguéssim modificar la longitud del vector, o guardar els valors en posicions diferents, només hauríem de modificar la constant al principi del codi, en comptes d'haver de modificar totes les línies on apareixen aquests valors numèrics.

Pseudocodi correcte:

```
const
    NUM_SEATS1: integer = 34;
    NUM_RIDES: integer = 3;
    A1: integer = 1;
    A2: integer = 2;
    A3: integer = 3;
end const

var
    emptySeats[NUM_RIDES]: vector of integer;
end var

{input values}
writeString("EMPTY SEATS  ");
writeString(NAME_RIDE1);
writeString("  >> ");
emptySeats[A1] := readInteger();
```

3.8.8 Constants: declaració de constants mal ubicada

Codi incorrecte:

```
#include <stdio.h>

int main(int argc, char **argv) {
    #define MAX_LEN 15
    char brand[MAX_LEN];
    return 0;
}
```

Les constants s'han de declarar **al principi del bloc de codi**, i **fora** de qualsevol funció (sigui el `main` o una altra), ja que igual que els tipus de dades, les constants s'entenen globals a tot el programa. No és correcte obrir un bloc de declaració de constants dins un bloc central de codi, i encara menys obrir diversos blocs de constants a mesura que les necessitem.

Codi correcte:

```c
#include <stdio.h>
#define MAX_LEN 15

int main(int argc, char **argv) {
    char brand[MAX_LEN];
    /* ... */
    return 0;
}
```

3.8.9 Strings: comparació directa

Codi incorrecte:

```c
#include <stdio.h>
#define MAX_LEN 15

int main(int argc, char **argv) {
    char name1[MAX_LEN];
    char name2[MAX_LEN];
    if (name1 == name2) {
        /* ... */
    }
    return 0;
}
```

En C les cadenes de caràcters (així com la resta de vectors) **no** es poden comparar directament amb l'operador `==`. En el seu lloc tenim dues opcions:

- En el cas dels vectors, es poden comparar element a element, de forma individual
- En el cas dels strings, el C disposa de funcions específiques, com ara `strcmp()`.

Codi correcte:

```c
#include <stdio.h>
#define MAX_LEN 15

int main(int argc, char **argv) {
    char name1[MAX_LEN];
    char name2[MAX_LEN];
    if (strcmp(name1, name2) == 0) {
        /* ... */
    }
    return 0;
}
```

Chapter 4

PAC04

4.1 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
      { abans de finalitzar tot el bloc d'instruccions que executa, ja que en cas cont
      { 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
         * valor sempre seria de i == 0
         */
        i = i+1;
    }
    return 0;
}
```

4.2 Entrada contínua 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:

```
#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 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"*.

4.3 Com tractar valors en múltiples vectors

Imaginem que volem introduir per teclat una sèrie de dades dels treballadors de la nostra empresa: dni, dies d'antiguitat i sou brut anual. Aquestes dades les introduïrem en tres vectors diferents: un pels DNI, l'altre per l'antiguitat i l'últim pel sou brut anual. El valor del sou net menual es calcularà a partir del brut i es desarà també en un vector.

El programa ha de demanar per teclat les dades de 5 empleats, i al finalitzar mostrarà els valors per pantalla de la següent forma:

```
>> empleat: 39284019x
   antiguitat (dies): 784
   brut anual (€): 36874.78
   net mensual (€): 1659.36

>> empleat: 31214557m
   antiguitat (dies): 128
   brut anual (€): 20015.30
   net mensual (€): 1086.54
```

El sou net mensual es calcula aplicant la següent retenció, i posteriorment dividint per 14 pagues:

sou brut	retenció
sou <12450.0€	19.0%
12450.0€ <= sou < 20200.0€	24.0%
20200.0€ <= sou < 35200.0€	30.0%
35200.0€ <= sou < 60000.0€	37.0%
sou > 60000.0€	45.0%

L'algorisme podria ser el següent:

```
const
    MAX_ELEMS: integer = 5;
    TRAM1: float = 12450.0;
    RETENCIO1: float = 19.0;
    TRAM2: float = 20200.0;
    RETENCIO2: float = 24.0;
    TRAM3: float = 35200.0;
    RETENCIO3: float = 30.0;
    TRAM4: float = 60000.0;
    RETENCIO4: float = 37.0;
    RETENCIO5: float = 45.0;
    NUM_PAGUES: integer = 14;
end const

algorithm vectorsDeVehicles

    var
        vDni: vector[MAX_ELEMS] of string;
        vAntiguitat: vector[MAX_ELEMS] of integer;
        vBrutAnual: vector[MAX_ELEMS] of real;
        vNetMensual: vector[MAX_ELEMS] of real;
        i: integer;
    end var

    { La lectura de dades des del canal d'entrada }
    { S'utilitza un únic índex, 'i', per recórrer tots els vectors }
    for i := 1 to MAX_ELEMS do
        writeString("dades empleat num. ");
        writeInteger(i);
        writeString(":");
        writeString(">> dni : ");
        vDni[i] := readString();
```

```
        writeString(">> antiguitat : ");
        vAntiguitat[i] := readInteger();
        writeString(">> brut anual : ");
        vBrutAnual[i] := readReal();

        { Càlcul del sou net mensual }
        if (vBrutAnual[i] < TRAM1) then
            vNetMensual[i] := (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO1/100)) / NUM_PA
        else
            if (vBrutAnual[i] < TRAM2) then
                vNetMensual[i] := (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO2/100)) / NU
            else
                if (vBrutAnual[i] < TRAM3) then
                    vNetMensual[i] := (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO3/100)) /
                else
                    if (vBrutAnual[i] < TRAM4) then
                        vNetMensual[i] := (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO4/10
                    else
                        vNetMensual[i] := (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO5/10
                    end if
                end if
            end if
        end if
    end for

    { Es mostren les dades pel canal de sortida }
    { S'utilitza un únic índex, 'i', per recórrer tots els vectors }
    for i := 1 to MAX_ELEMS do
        writeString(">> empleat: ");
        writeString(vDni[i]);
        writeString("   antiguitat (dies): ");
        writeInteger(vAntiguitat[i]);
        writeString("   brut anual (€): ");
        writeInteger(vBrutAnual[i]);
        writeString("   net mensual (€): ");
        writeInteger(vNetMensual[i]);
    end for

end algorithm
```

Com ho podem implementar en C? Una possible solució seria la següent:

```c
#include <stdio.h>

#define MAX_ELEMS 5
#define TRAM1 12450.0
#define RETENCIO1 19.0
#define TRAM2 20200.0
#define RETENCIO2 24.0
#define TRAM3 35200.0
#define RETENCIO3 30.0
#define TRAM4 60000.0
#define RETENCIO4 37.0
#define RETENCIO5 45.0
#define NUM_PAGUES 14
#define MAX_DNI 9+1

typedef char tDni[MAX_DNI];

int main(int argc, char **argv) {
    tDni vDni[MAX_ELEMS];
    int vAntiguitat[MAX_ELEMS];
    float vBrutAnual[MAX_ELEMS];
    float vNetMensual[MAX_ELEMS];
    int i;

    /* La lectura de dades des del canal d'entrada.
       S'utilitza un únic índex, i, per recórrer tots els vectors */
    for (i = 0; i < MAX_ELEMS; i++) {
        printf("dades empleat num. %d : \n", i);
        printf(">> dni : ");
        scanf("%s", vDni[i]);
        printf(">> antiguitat : ");
        scanf("%d", &vAntiguitat[i]);
        printf(">> brut anual : ");
        scanf("%f", &vBrutAnual[i]);

        /* Càlcul del sou net mensual */
        if (vBrutAnual[i] < TRAM1) {
            vNetMensual[i] = (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO1/100)) / NUM_PAGUES;
        } else {
            if (vBrutAnual[i] < TRAM2) {
                vNetMensual[i] = (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO2/100)) / NUM_PAGUES;
            } else {
                if (vBrutAnual[i] < TRAM3) {
                    vNetMensual[i] = (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO3/100)) / NUM_PAGUES
                } else {
```

```
                    if (vBrutAnual[i] < TRAM4) {
                        vNetMensual[i] = (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO4/100)
                    } else {
                        vNetMensual[i] = (vBrutAnual[i] - (vBrutAnual[i]*RETENCIO5/100)
                    }
                }
            }
        }
    }

    /* Es mostren les dades pel canal de sortida.
       S'utilitza un únic índex, i, per recórrer tots els vectors */
    for (i = 0; i < MAX_ELEMS; i++) {
        printf("\n>> empleat: %s \n", vDni[i]);
        printf("   antiguitat (dies): %d \n", vAntiguitat[i]);
        printf("   brut anual (€): %.2f \n", vBrutAnual[i]);
        printf("   net mensual (€): %.2f \n", vNetMensual[i]);
    }
    return 0;
}
```

Dins del bucle utilitzem la variable `i` com a índex per anar recorrent tots els vectors alhora.

En els dos casos la inserció dels valors en els vectors la fem de la mateixa forma: utilitzem l'índex `i` per determinar la posició del vector on ubicarem els valors:

```
/* ... */
scanf("%s", vDni[i]);
/* ... */
scanf("%d", &vAntiguitat[i]);
/* ... */
scanf("%f", &vBrutAnual[i]);
```

Al final de l'exemple mostrem tots els empleats introduïts mitjançant un segon bucle, mostrant totes les dades introduïdes anteriorment i les calculades.

4.4 Definició chars vs strings

En el llenguatge C, els caràcters es defineixen sempre amb cometa simple `'`, mentre que pels string s'utilitza la cometa doble `"`.

Exemple:

```c
#include <stdio.h>

int main(int argc, char **argv) {
    /* Assignació de valor a un string amb cometa doble */
    char salutacio[]= "Hi World";
    /* Assignació de valor a un char amb cometa simple */
    char exclamacio = '!';

    printf("%s %c\n", salutacio, exclamacio);
    return 0;
}
```

4.5 Exemple: mitjanaPes

Imaginem que volem fer un programa que ens ajudi a calcular el nostre pes promig setmanal. Afegirem per teclat les mesures diàries del nostre pes al programa i, en cas de trobar algun valor incoherent l'obviarà i el tornarà a demanar.

En llenguatge algorísmic ho podem implementar de la següent forma:

```
const
    NUM_DIES: integer = 7;
    PES_MIN: real = 50.0;
    PES_MAX: real = 110.0;
end const

algorithm mitjanaPes

    var
        i: integer;    { Comptador que utilitzarà el nostre programa }
        aux: real;   { Variable auxiliar per la lectura de pesos }
        vectorPesos: vector[NUM_DIES] of real;   { Pes en Kg: 79.5, ... }
        sumaPesos: real;
    end var

    i := 1;
```

```
    sumaPesos := 0;

    { Es llegeix des de teclat els pesos diaris, un a un }
    while i   NUM_DIES do

        { Llegim un valor des del canal standard d'entrada }
        writeString("Introdueix pes (Kg.) : ");
        aux := readReal();

        { A continuació cal revisar que aquest valor sigui coherent. Prenem com a valor
        "possibles" aquells que estiguin entre PES_MIN i PES_MAX. Fixeu-vos que la lect
        es desa temporalment a la variable aux: quan haguem validat que contingui un va
        vàlid, l'afegirem dins del vector de pesos }

        if (PES_MIN   aux) and (aux   PES_MAX) then
            { En aquest punt, la variable aux conté un valor correcte, amb el que ja el
            afegir al vector de pesos }
            vectorPesos[i] := aux;

            { El següent punt és molt important: com que ja hem afegit el pes correcte
            incrementarem la variable 'i', la qual ens serveix per accedir a una nova p
            del vector a la següent iteració del bucle }
            i = i + 1;

        else
            { En cas contrari, el pes es incorrecte amb el que mostrem el corresponent
            missatge d'error }
            writeString("Pes incorrecte!");

            { Important: aquí no incrementem la variable i, ja que el valor no és corre
            que en la següent iteració del bucle, es continuï intentant afegir el valor
            de la posició 'i' del bucle }

        end if

        { En aquest punt tenim el vectorPesos que té 7 (=NUM_DIES) pesos vàlids }

        { Ara utilitzarem un segon bucle per recórrer tots els valors del vector i fer
        demanat: la mitjana de tots ells. Si haguéssim volgut, ho hauríem pogut fer tot
        en un únic bucle, però he preferit separar-ho perquè quedi més clar què es fa
        dins de cadascun dels dos bucles}

        for  i := 1 to NUM_DIES do

            { Dins de la variable sumaPesos hi anem sumant cadascun dels pesos
            de les posicions del vector }
```

```
            sumaPesos := sumaPesos + vectorPesos[i];

        end for

        { Per calcular la mitjana, únicament ens falta dividir el valor sumaPesos pel
        número total de pesos introduïts o, el que és el mateix, NUM_DIES }
        writeString("La mitjana setmanal del pes és : ");
        writeReal(sumaPesos / NUM_DIES);

end algorithm
```

Una possible implementació en llenguatge C d'aquest algorisme pot ser la següent:

```c
#include <stdio.h>

#define NUM_DIES 7
#define PES_MIN 50.0
#define PES_MAX 110.0

int main (int argc, char **argv){
    float vectorPesos[NUM_DIES];
    int i;
    float aux;
    float sumaPesos;

    i = 0; /* Important! en llenguatge C els vectors comencen pel 0 */
    sumaPesos = 0;

    /* Primer bucle: introducció i validació de dades */
    while (i<NUM_DIES) {

        printf("Introdueix pes (Kg.) : ");
        scanf("%f", &aux);

        if (PES_MIN <= aux && aux <= PES_MAX) {
            vectorPesos[i] = aux;
            i = i+1;
        } else {
            printf("Pes incorrecte!\n");
        }
    }
```

```
    /* Segon bucle: càlcul intermig per la mitjana de pesos */
    for (i=0; i<NUM_DIES; i++) {
        sumaPesos = sumaPesos + vectorPesos[i];
    }

    printf("La mitjana setmanal del pes és : %.2f", sumaPesos/NUM_DIES);
}
```

4.6 Frequently Made Mistakes

4.6.1 Estructura iterativa: for vs while

Aquest no és un error sintàctic o semàntic, sinó un error de disseny freqüent.

Codi incorrecte:

```
#include <stdio.h>
#include <stdbool.h>
#define NUM 10

int main(int argc, char **argv) {
    int i;
    char password[NUM] = {'a', '1', 'h', '\0'};

    for (i = 0; password[i] != '\0'; i++){
        /* ... */
    }
    return 0;
}
```

En el codi anterior, sembla que volem recórrer un vector de caràcters element a element, per executar alguna acció. Per fer-ho, decidim utilitzar un bucle i una condició final: les iteracions s'aturaran en el moment en què ens trobem amb el caràcter especial `'\0'`, finalitzador de cadena de caràcters.

El problema és que el bucle `for` no és l'estructura més indicada per resoldre aquest algorisme, ja que està pensat per repetir un bloc de codi un nombre de vegades predeterminat, basat gairebé sempre en un índex (`i`) que s'ha d'incrementar/decrementar des d'un valor inicial a un de final. En aquest cas, la condició final no té res a veure amb el valor numèric de l'índex, i per tant és molt millor utilitzar el bucle `while`.

Codi correcte:

```c
#include <stdio.h>
#include <stdbool.h>
#define NUM 10

int main(int argc, char **argv) {
    int i;
    char password[NUM] = {'a', '1', 'h', '\0'};
    i = 0;
    while (password[i] != '\0'){
        /* ... */
    i++;
    }
    return 0;
}
```

És important recordar que en el cas de l'estructura `while`, l'índex `i` **no s'actualitza automàticament**: cal fer-ho manualment abans de tancar el bloc.

Chapter 5

PAC05

5.1 strcmp()

En què es basa `strcmp()` per decidir que, per exemple, la lletra `'O'` és més gran que la lletra `'A'`?

La resposta la tenim en el codi 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 volem consultar la taula ASCII per internet, la podem generar nosaltres mateixos 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);
    }
    return 0;
}
```

5.2 scanf()

Quan utilitzem `scanf()` fins ara sempre li hem passat el nom de la variable precedit per `&`. Això significa que realmente li estem passant la posició de la memòria on resideix la variable facilitada.

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).

A continuació s'adjunta un exemple amb tots aquests conceptes:

```c
#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;
}
```

5.3 El finalitzador '\0' i strcmp()

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(int argc, char **argv) {
    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));
    return 0;
}
```

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 tenim 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.

5.4 El finalitzador '\0' i strlen()

A continuació s'exposa un exemple en el qual es mostra la importància del finalitzador `'\0'` en la funció `strlen()` de C, la qual ens retorna la mida d'una cadena de caràcters :

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

#define MAX_LLETRES 8+1

int main(int argc, char **argv) {
    char nom[MAX_LLETRES];
    int numLletres;

    printf("Introdueix un nom: ");
    scanf("%s", nom);

    /* Exemple: si en aquest punt hem teclejat el nom
     * Quim, dins de la cadena de caràcters nom[]
     * tindrem les següents dades:
     *
     * nom[0] = 'Q'
     * nom[1] = 'u'
     * nom[2] = 'i'
     * nom[3] = 'm'
     * nom[4] = '\0' (finalitzador de l'string)
     * nom[5] = valor aleatori
     * nom[6] = valor aleatori
     * nom[7] = valor aleatori
     * nom[8] = valor aleatori
     *
     * El finalitzador '\0' s'afegeix automàticament
     * en llegir un string per teclat amb scanf.
     *
     * La comanda strlen(...) va recorrent l'string posició
     * a posició per saber la seva longitud. Quan finalitza
     * aquest recorregut? hi ha dues opcions possibles:
     *
     * - quan troba el finalitzador '\0'
     * - quan arriba a la darrera posició de l'string
     *
```

```
     * Per tant, si com a nom hem entrat Quim, el valor que
     * retornarà strlen(...) serà 4.
     */

    numLletres = strlen(nom);

    printf("El nom \"%s\" té %d lletres.\n", nom, numLletres);

    /* Què passa si sobreescrivim el finalitzador '\0' amb
     * un caràcter qualsevol? per exemple 'X'
     */
    printf("\nSobreescrivim el finalitzador '\\0'.");
    nom[numLletres] = 'X';

    /* Tornem a calcular la longitud de l'string */
    numLletres = strlen(nom);

    /* Per quin motiu ara ha canviat la longitud de la
     * variable nom, si no l'hem tornat a redefinir?
     */
    printf("\nAra el nom \"%s\" té %d lletres.\n", nom, numLletres);

    return 0;
}
```

Si s'executa el programa, la sortida obtinguda serà similar a la següent:

```
Introdueix un nom: Quim
El nom "Quim" té 4 lletres.

Sobreescrivim el finalitzador '\0'.
Ara el nom "QuimX ! " té 9 lletres.
```

5.5 Exemple: comparacioStrings

La forma com comparem strings amb llenguatge algorísmic és diferent que no en llenguatge C:

- Llenguatge algorísmic: la comparació entre strings es fa amb `=`.
- Llenguatge C: la comparació entre strings es fa amb la funció `strcmp()`, la qual realitza una comparació caràcter a caràcter de les dues 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.

Un exemple on realitza una comparació de dos strings pot ser el següent:

```
algorithm comparacioStrings

    var
        cadena1: string;
        cadena2: string;
    end var

    cadena1:= "UOC";
    cadena2:= "UAB";

    if (cadena1 = cadena2) then
        writeString(cadena1);
        writeString(" = ");
        writeString(cadena2);
    else
        if (cadena1 > cadena2) then
            writeString(cadena1);
            writeString(" > ");
            writeString(cadena2);
        else
            writeString(cadena1);
            writeString(" < ");
            writeString(cadena2);
        end if
    end if

end algorithm
```

En llenguatge C, la comparació caràcter a caràcter entre els string `"UOC"` i `"UAB"` 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.

```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", resultatCom
    } else if (resultatComparacio == -1) {
            printf("El resultat %d significa que l'string \"%s\" < string \"%s\"\n", resultatComp
    } else if (resultatComparacio == 1) {
            printf("El resultat %d significa que l'string \"%s\" > string \"%s\"\n", resultatComp
    }
    return 0;
}
```

El resultat de l'execució del programa en C és:

```
Comparació strings "UOC" i "UAB" = 1
El resultat 1 significa que l'string "UOC" > string "UAB"
```

5.6 Exemple: nòmines

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[ma
    printf("\nNòmina més baixa : %.2f € (%s)", vEmpleats[minNomina].nomina, vEmpleats[m
```

```
    return 0;
}
```

Com es pot veure, s'utilitza un vector de `tEmpleat` de forma que, donada una longitud màxima del vector, anirem introduint els `tEmpleat` un a un dins d'ell. Una vegada fet, tornem 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.

5.7 Exemple: brisca

Exemple amb diferents recorreguts realitzat sobre tuples guardades en un vector, utilitzant el joc de cartes de la brisca. Per tal d'explicar millor el plantejament de cada implementació, s'han afegit comentaris detallats dins del codi.

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

/* Punt de partida: sabem jugar perfectament
 * a la brisca (https://ca.wikipedia.org/wiki/Brisca),
 * però en canvi som un desastre havent de comptar
 * la puntuació de totes les cartes guanyades. Per
 * aquest motiu volem fer un programa que ens ajudi
 * en aquesta tasca (i també per practicar diferents
 * iteracions amb bucles). El programa demanarà per
 * teclat carta a carta i finalitzarà quan s'introdueixi
 * un tipus de coll diferent dels definits.
 * Haurà de generar les següents sortides:
 * 1. Llistar totes les cartes introduïdes.
 * 2. Mostrar la carta amb una puntuació més alta.
 * 3. Mostrar la puntuació total aconseguida.
 * 4. Comparar parelles de cartes (1a vs 2a, 2a vs 3a,
 *    3a vs 4a,...) i indicar si són del mateix coll.
 */

#define MAX_NUM_CARTES 48
#define NUM_VALORS 13
```

```c
#define MAX_NUM_COLLS 4

typedef enum {MONEDES, COPES, BASTOS, ESPASES, FINALITZAR} tColl;
typedef struct {
    tColl coll;
    int numero;
    int valor;
} tCarta;

int main(int argc, char **argv) {
    tCarta cartaAux;
    tCarta cartaDeMajorValor;
    tCarta vCartes[MAX_NUM_CARTES];
    bool isFinalitzat;
    int puntuacio;
    int i, j;
    bool isMateixColl;

    /* S'inicialitza el vector vValors amb les
     * puntuacions de totes les cartes. El nom
     * de la carta fa d'índex del vector, i el
     * valor desat en aquella posició és la seva
     * puntuació. Exemple:
     * - una carta amb un 1 retornarà 11 punts,
     *   corresponents al valor de la posició 1.
     * - una carta amb un 5 retornarà 0 punts,
     *   corresponents al valor de la posició 5.
     */
    int vValors[NUM_VALORS] = {0, 11, 0, 10, 0, 0, 0, 0, 0, 0, 2, 3, 4};

    /* El següent vector ens ajudarà a mostrar
     * per pantalla de forma automàtica la descripció
     * associada a cadascun dels valors definits a
     * l'enumeratiu tColl
     */
    char* vNoms[MAX_NUM_COLLS] = {"Monedes", "Copes", "Bastos", "Espases"};

    /* La variable booleana s'utilitzarà per
     * finalitzar el primer bucle d'introducció
     * de cartes des de teclat
     */
    isFinalitzat = false;

    i = 0;
    j = 0;
```

```
    puntuacio = 0;

    printf("Tipus carta: 0=MONEDES, 1=COPES, 2=BASTOS, 3=ESPASES.\n");

    while (!isFinalitzat) {
        printf("\nIntrodueix tipus carta: ");
        scanf("%u", &cartaAux.coll);

        /* Si el coll introduit no correspon a cap dels
         * quatre definits, significa que no es vol
         * entrar cap més carta des de teclat
         */
        if (cartaAux.coll != MONEDES && cartaAux.coll != COPES
            && cartaAux.coll != BASTOS && cartaAux.coll != ESPASES) {
            isFinalitzat = true;

        } else {
            printf("Introdueix número carta: ");
            scanf("%d", &cartaAux.numero);

            /* Es valida que el número de la carta
             * introduïda sigui vàlid: valor comprès
             * entre 1 i 12
             */
            if (cartaAux.numero < 1 || cartaAux.numero > 12) {
                printf("Error: número de la carta incorrecte!\n");

            } else {
                /* Es calcula el valor de la carta i
                 * s'assigna al camp valor de la tupla
                 * cartaAux (de tipus tCarta)
                 */
                cartaAux.valor = vValors[cartaAux.numero];
                vCartes[i] = cartaAux;

                /* La variable 'i' contindrà el número
                 * de cartes vàlides introduïdes per
                 * teclat.
                 */
                i = i + 1;
            }
        }
    }

    /* Sortida 1:
```

```
     * Recorrem el vector de cartes per mostrar
     * per pantalla tots els elements (cartes)
     * que conté.
     */
    printf("\nRelació de cartes introduïdes: \n");
    for (j = 0; j < i; j++) {
        printf("\t%d de %s (%d punts) \n", vCartes[j].numero, vNoms[vCartes[j].coll], v
    }

    /* Sortida 2:
     * Recorrem el vector de cartes i busquem
     * la que té una major puntuació. En cas
     * d'empat, mostrem l'última introduïda.
     */

    /* Inicialitzem el valor de cartaDeMajorValor
     * a valor = 0
     */
    cartaDeMajorValor.valor = 0;
    printf("\nCarta de major puntuació: \n");
    for (j = 0; j < i; j++) {
        /* En cada iteració del bucle ens assegurem
         * que cartaDeMajorValor sigui la carta
         * amb un valor més gran. Comparem la carta
         * que estem tractant actualment amb la carta
         * que fins ara hem trobat de major valor.
         */
        if (vCartes[j].valor >= cartaDeMajorValor.valor) {
            cartaDeMajorValor.coll = vCartes[j].coll;
            cartaDeMajorValor.numero = vCartes[j].numero;
            cartaDeMajorValor.valor = vCartes[j].valor;
        }
    }
    printf("\t%d de %s (%d punts) \n", cartaDeMajorValor.numero, vNoms[cartaDeMajorVal

    /* Sortida 3:
     * Recorrem el vector de cartes i anem sumant
     * tots els valors, per tal d'obtenir la
     * puntuació total.
     */
    printf("\nPuntuació total: \n");
    for (j = 0; j < i; j++) {
        puntuacio = puntuacio + vCartes[j].valor;
    }
    printf("\t%d punts \n", puntuacio);
```

```
    /* Sortida 4:
     * Recorrem el vector de cartes i anem comparant
     * parelles de carta: la que estem tractant amb
     * la que ve a continuació. Mostrem per pantalla
     * si són del mateix coll o no.
     */
    printf("\nComparació de cartes: \n");

    /* Important: en aquest recorregut del vector
     * de cartes anem comparant la carta de la
     * posició actual j amb la carta que està
     * a la següent posició j+1. Això significa
     * que el límit de les iteracions del
     * bucle passa a ser i-1 (un element abans
     * del final), ja que quan tractem aquest
     * element el compararem amb el següent,
     * que és l'últim del vector. Si en comptes
     * de tenir i-1 tinguéssim només i, en
     * la darrera iteració donaria error, ja
     * que s'estaria accedint a una posició
     * incorrecta del vector de cartes (i+1).
     */
    for (j = 0; j < i-1; j++) {

        /* Per comparar la carta de la posició actual
         * amb la que hi ha a la següent posició,
         * utilitzem els índex j i j+1 respectivament.
         */
        isMateixColl = (vCartes[j].coll == vCartes[j+1].coll);
        printf("\t%d de %s vs %d de %s: ", vCartes[j].numero, vNoms[vCartes[j].coll], vCartes[j+1
        if (isMateixColl) {
            printf("són del mateix coll!\n");
        } else {
            printf("són de colls diferents!\n");
        }
    }
    return 0;
}
```

Un exemple d'execució pot ser el següent:

```
Tipus carta: 0=MONEDES, 1=COPES, 2=BASTOS, 3=ESPASES.
```

```
Introdueix tipus carta: 0
Introdueix número carta: 1

Introdueix tipus carta: 0
Introdueix número carta: 10

Introdueix tipus carta: 0
Introdueix número carta: 5

Introdueix tipus carta: 2
Introdueix número carta: 10

Introdueix tipus carta: 2
Introdueix número carta: 1

Introdueix tipus carta: 3
Introdueix número carta: 4

Introdueix tipus carta: 9

Relació de cartes introduïdes:
    1 de Monedes (11 punts)
    10 de Monedes (2 punts)
    5 de Monedes (0 punts)
    10 de Bastos (2 punts)
    1 de Bastos (11 punts)
    4 de Espases (0 punts)

Carta de major puntuació:
    1 de Bastos (11 punts)

Puntuació total:
    26 punts

Comparació de cartes:
    1 de Monedes vs 10 de Monedes: són del mateix coll!
    10 de Monedes vs 5 de Monedes: són del mateix coll!
    5 de Monedes vs 10 de Bastos: són de colls diferents!
    10 de Bastos vs 1 de Bastos: són del mateix coll!
    1 de Bastos vs 4 de Espases: són de colls diferents!
```

5.8 Frequently Made Mistakes

5.8.1 Definició de tipus: definició de tuples mal ubicada

Codi incorrecte:

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

int main(int argc, char **argv) {
    typedef struct{
        int id;
        char brand[MAX_LEN];
        float price;
    } tHotel;

    tHotel myHotel;
    /* ... */
    return 0;
}
```

Els tipus de dades s'han de declarar **al principi del bloc de codi**, i fora de qualsevol funció (sigui el `main` o una altra), ja que els tipus són globals a tot el programa, i les variables associades als tipus es declaren posteriorment en cada una de les funcions on es necessiten.

Les tuples (`structs`) no en són una excepció, i per tant no és correcte obrir un bloc de definició de tuples dins un bloc central de codi (i encara menys obrir diversos blocs de definició de tuples a mesura que els necessitem).

Codi correcte:

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

typedef struct{
    int id;
    char brand[MAX_LEN];
    float price;
} tHotel;
```

```c
int main(int argc, char **argv) {
    tHotel myHotel;
    /* ... */
    return 0;
}
```

5.8.2 Definició de tipus: definició mal ubicada

Codi incorrecte:

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

int main(int argc, char **argv) {
    typedef enum {BUDGET, INN, RESORT} tTypeHotel;
    typedef enum {STANDARD, SUITE} tTypeRoom;

    tTypeHotel myHotelType = BUDGET;
    tTypeRoom myRoomType = STANDARD;
    /* ... */
    return 0;
}
```

Els tipus de dades s'han de declarar **al principi del bloc de codi**, i fora de qualsevol funció (sigui el `main` o una altra), ja que els tipus són globals a tot el programa, i les variables associades als tipus es declaren posteriorment en cada una de les funcions on es necessiten. No és correcte obrir un bloc de definició de tipus dins un bloc central de codi (i encara menys obrir diversos blocs de definició de tipus a mesura que els necessitem).

Codi correcte:

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

typedef enum {BUDGET, INN, RESORT} tTypeHotel;
typedef enum {STANDARD, SUITE} tTypeRoom;

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

```
    tTypeHotel myHotelType = BUDGET;
    tTypeRoom myRoomType = STANDARD;
    /* ... */
    return 0;
}
```

Chapter 6

PAC06

6.1 Diferències entre funcions i accions

A continuació s'expliquen les diferències entre una **funció** i una **acció**, quins tipus de paràmetres utilitzen, com s'implementen en llenguatge C, i finalment què passa quan un paràmetre és una tupla. És important que es vagin consolidant tots aquests conceptes.

Les principals diferències són:

	Funcions	Accions
Retornen un valor?	sí	no
Tipus de paràmetres	entrada (in)	entrada (in), sortida (out), entrada/sortida (inout)

El retorn de valor de les funcions permet que aquest es pugui assignar a una variable, cosa que no es pot fer amb les accions.

Per exemple, imaginem que volem implementar en llenguatge C la funció `suma()`; una possible implementació podria ser:

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

```c
/* Predeclaració de la funció */
int suma(int n1, int n2);

int main(int argc, char **argv) {
    int num1 = 3;
    int num2 = 2;
    int resultat = 0;
    printf("Valor de num1 = %d\n", num1);
    printf("Valor de num2 = %d\n", num2);
    printf("Valor de resultat = %d\n", resultat);
    printf(">> Inici execució funció\n");
    resultat = suma(num1, num2);
    printf("Suma = %d\n", resultat);
    printf(">> Fi execució funció\n");
    printf("Valor de resultat = %d\n", resultat);
    return 0;
}

/* Implementació de la funció */
int suma(int n1, int n2) {
    return (n1+n2);
}
```

L'execució generarà la següent sortida:

```
Valor de num1 = 3
Valor de num2 = 2
Valor de resultat = 0
>> Inici execució funció
Suma = 5
>> Fi execució funció
Valor de resultat = 5
```

Com es pot veure, el valor de retorn de la funció `suma()` l'assignem a la variable resultat.

En una funció, els paràmetres passats sempre seran **d'entrada** (in): això significa que dins de la funció únicament seran valors de consulta, no els modificarem per res.

Per entendre bé com s'implementa una acció, relacionarem exemples similars amb els diferents tipus de paràmetres que pot tenir una acció: **entrada** (in), **sortida** (out) i **entrada/sortida** (inout).

6.1.1 Paràmetres d'entrada (in)

Són aquells paràmetres que es passen a una **acció** i dels quals únicament utilitzarem el seu contingut. Això significa que treballarem amb ells en *mode lectura*: obtindrem els seus valors per tal de realitzar càlculs, però mai modificarem el seu contingut. Exemple:

```c
#include <stdio.h>

/* Predeclaració de l'acció */
void suma(int n1, int n2);

int main(int argc, char **argv) {
    int num1 = 3;
    int num2 = 2;
    printf("Valor de num1 = %d\n", num1);
    printf("Valor de num2 = %d\n", num2);
    printf(">> Inici execució acció\n");
    suma(num1, num2);
    printf(">> Fi execució acció\n");
    printf("Valor de num1 = %d\n", num1);
    printf("Valor de num2 = %d\n", num2);
    return 0;
}

/* Implementació de l'acció */
void suma(int n1, int n2) {
    int resultat = n1 + n2;
    printf("Suma = %d\n", resultat);
}
```

L'execució generarà la següent sortida:

```
Valor de num1 = 3
Valor de num2 = 2
>> Inici execució acció
Suma = 5
>> Fi execució acció
Valor de num1 = 3
Valor de num2 = 2
```

Com es pot observar, ni `num1` ni `num2` han modificat el seu valor després de l'execució de l'acció: son **paràmetres d'entrada**.

Aquest tipus de paràmetre també es referencia com a paràmetre per valor, o pas per valor, ja que el que estem passant és un valor (no pas un punter).

6.1.2 Paràmetres de sortida (out)

A diferencia dels paràmetres d'entrada, els de sortida s'utilitzen únicament per guardar valors. Poden contenir qualsevol valor inicial, que aquest no serà utilitzat dins de l'acció. Una vegada realitzats tots els càlculs de l'acció, el resultat final es guardarà en el paràmetre de sortida. Exemple:

```c
#include <stdio.h>

/* Predeclaració de l'acció */
void suma(int n1, int n2, int *res);

int main(int argc, char **argv) {
    int num1 = 3;
    int num2 = 2;
    int resultat = 0;
    int *pResultat = &resultat;
    printf("Valor de num1 = %d\n", num1);
    printf("Valor de num2 = %d\n", num2);
    printf("Valor de resultat = %d\n", resultat);
    printf(">> Inici execució acció\n");
    suma(num1, num2, pResultat);
    printf("Suma = %d\n", resultat);
    printf(">> Fi execució acció\n");
    printf("Valor de resultat = %d\n", resultat);
    return 0;
}

/* Implementació de l'acció */
void suma(int n1, int n2, int *res) {
    *res = n1 + n2;
}
```

L'execució generarà la següent sortida:

```
Valor de num1 = 3
Valor de num2 = 2
Valor de resultat = 0
>> Inici execució acció
Suma = 5
>> Fi execució acció
Valor de resultat = 5
```

De moment ignorarem que es tracta d'un punter: independientement del valor que tingui res al pasar-se per paràmetre, quan finalitzi l'acció contindrà la suma dels altres dos paràmetres d'entrada `num1` i `num2`.

El valor resultat ha canviat, de `0` a `5`. Es considera un **paràmetre de sortida** perquè, independientement del valor inicial que tingui aquest, no s'utilitza per res i en finalitzar l'acció contindrà el resultat de l'operació `suma()`.

Ara sí comentem el fet d'utilitzar el punter: la manera que tenim a C per modificar una variable definida fora d'una acció des de dins de la mateixa és treballant precisament amb la seva direcció de memòria.

Aquest tipus de paràmetre també es referencia com a paràmetre per referència, o pas per referència, ja que passem un punter.

6.1.3 Paràmetres d'entrada/sortida (inout)

Aquest tipus de paràmetre és una suma dels dos comportaments anteriors: d'una banda el seu valor importa a l'hora de realitzar els càlculs de l'acció, i a la vegada en finalitzar l'execució de l'acció tindrà un valor diferent, també significatiu per tractar-se del resultat final del càlcul. Exemple:

```c
#include <stdio.h>

/* Predeclaració de l'acció */
void suma(int *pN1, int n2);

int main(int argc, char **argv) {
    int num1 = 3;
    int num2 = 2;
    int *pNum1 = &num1;
    printf("Valor de num1 = %d\n", num1);
```

```
    printf("Valor de num2 = %d\n", num2);
    printf(">> Inici execució acció\n");
    suma(pNum1, num2);
    printf("Suma = %d\n", num1);
    printf(">> Fi execució acció\n");
    printf("Valor de num1 = %d\n", num1);
    return 0;
}

/* Implementació de l'acció */
void suma(int *pN1, int n2) {
    *pN1 = *pN1 + n2;
}
```

L'execució generarà la següent sortida:

```
Valor de num1 = 3
Valor de num2 = 2
>> Inici execució acció
Suma = 5
>> Fi execució acció
Valor de num1 = 5
```

En aquest cas s'ha realitzat la suma com a `num1 = num1 + num2` : el resultat de la suma de les dues variables es guarda a la primera d'elles. Així doncs el valor de la variable `num1` importa tant a l'entrada com a la sortida de l'acció, amb el que es tracta d'un **paràmetre d'entrada/sortida**.

Aquest tipus de paràmetre també es referencia com a paràmetre per referència, o pas per referència, ja que passem un punter.

6.1.4 Tuples com a paràmetres

Quan un paràmetre d'una acció/funció és una tupla , la forma com accedirem als seus atributs dins de l'acció/funció variarà en funció del tipus de paràmetre:

- Paràmetres d'**entrada**: l'accessor als atributs és el . (un punt). Per tant dins de la funció/acció, accedirem als atributs de la tupla passada per paràmetre de la següent forma: `nomTupla.nomAtribut`.
- Paràmetres de **sortida** i d'**entrada/sortida**: en aquest cas l'accessor als atributs de la tupla és `->`. Així, dins de l'acció podrem fer referència als atributs de la tupla passada per paràmetre de la forma: `nomTupla->nomAtribut`.

6.2 Exemple: ús d'accions

A continuació s'adjunta un exemple inventat de com es poden definir accions que permetin modificar els atributs d'una tupla passada com a punter. Dins del codi hi ha comentaris addicionals per tal que quedi 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, fora de l'àmbit
     * de l'acció.
     * 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);
}
```

Un exemple d'execució seria:

```
Nom empleat : Laura
Nòmina empleat : 1850.32

Dades de l'empleat:
    Nom: Laura
    Nòmina: 1850.32 €
```

6.3 Exemple: ús de funcions

Si necessitem un mètode que com a resultat retorni un string, en comptes d'utilitzar una funció farem servir una acció amb un paràmetre de sortida o d'entrada/sortida. Les funcions les farem servir per retornar valors de **tipus primitiu** (enter, decimal, caràcter), i deixarem els **tipus compostos** (vectors, tuples, etc.) per les accions.

A continuació s'exposa un exemple perquè quedi més clar:

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

/* Programa que, donada una hora, indica
 * a quina part del dia correspon.
 * Per "part del dia" s'entèn: matinada,
 * matí, migdia, tarda, vespre i nit.
 * L'hora s'aconsegueix mitjançant una
 * funció, la qual retorna un valor
 * enter en format HHMM (on HH=hora i
 * MM=minut).
 * D'altra banda, el càlcul de la part
 * del dia es realitza amb una acció,
 * la qual rep dos paràmetres: un d'entrada
```

```c
 * corresponent a l'hora en format HHMM,
 * i un altre de sortida que contindrà
 * la part del dia (string) que correspon
 * a l'hora.
 */

#define MAX_CHARS 8+1

/* Predeclaració de funcions i accions */
void calcularPartDelDia(int hora, char *part);
int demanarHora();

/* Programa principal */
int main(int argc, char **argv) {
    int hora;
    char partDelDia[MAX_CHARS];
    hora = demanarHora();
    calcularPartDelDia(hora, partDelDia);
    printf("L'hora %d correspon a: %s \n", hora, partDelDia);
    return 0;
}

/* Implementació de funcions i accions */

/* Funció que retorna un enter, corresponent
 * a l'hora introduïda per teclat, en format
 * HHMM
 */
int demanarHora() {
    int hora;
    printf("Tecleja hora (format HHMM) : ");
    scanf("%d", &hora);
    return hora;
}

/* Acció que, donada una hora (paràmetre
 * d'entrada), calcula quina part del dia
 * correspon (paràmetre de sortida). La
 * part del dia és de tipus string. El
 * paràmetre de sortida ha de ser un
 * punter, per tal que des de dins de
 * l'acció es pugui modificar el valor
 * de la variable definida fora de
 * l'acció, dins del main.
 */
```

```c
void calcularPartDelDia(int hora, char *part) {

    /* Particionat horari extret de
     * https://www.parlament.cat/document/intrade/6698
     */

    /* 0 correspon a 0000 */
    /* 14 correspon a 0014 */
    if (hora >= 0 && hora <= 14) {
        strcpy(part, "nit");
    } else {
        /* 414 correspon a 0414 */
        if (hora > 14 && hora <= 414) {
            strcpy(part, "matinada");
        } else {
            /* 0500 correspon a 500 */
            if (hora > 414 && hora <= 1114) {
                strcpy(part, "matí");
            } else {
                if (hora > 1114 && hora <= 1414) {
                    strcpy(part, "migdia");
                } else {
                    if (hora > 1414 && hora <= 1814) {
                        strcpy(part, "tarda");
                    } else {
                        if (hora > 1814 && hora <= 2214) {
                            strcpy(part, "vespre");
                        } else {
                            if (hora > 2214 && hora <= 2359) {
                                strcpy(part, "vespre");
                            } else {
                                strcpy(part, "unknown!");
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Un exemple d'execució:

```
Tecleja hora (format HHMM) : 1906
L'hora 1906 correspon a: vespre
```

6.4 Exemple: nòmines

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, em
```

```
    return 0;
}
```

Fins aquest punt no hi ha res nou: utilitzem 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:

```c
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.

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 es pot 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, em
    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:

```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]);
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, em

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

    tEmpleat empleatCognom = cercaEmpleatPerCognom(vEmpleats, cognom);
    printf("\nDades de l'empleat: %s, %s -> %f", empleatCognom.cognom, empleatCognom.no
    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];
        }
    }
}
```

6.5 Exemple: pivotDefensiu

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

/* Rebem una petició d'un equip femení de bàsquet,
 * en el qual ens demanen un programa que els permeti
 * seleccionar la millor pivot defensiu d'entre una
 * sèrie de candidates.
 * La millor pivot defensiu és aquella que captura
 * més rebots; en cas d'empat, s'escollirà la que
 * faci més taps.
 * Caldrà implementar 3 accions i 1 funció:
 * - acció llegirJugadora(j): llegeix de teclat i
 *   guarda tots els atributs de la jugadora a la
 *   tupla j.
 * - acció mostrarJugadora(j): mostra per pantalla
 *   el valor dels atributs de la tupla j.
 * - acció copiarJugadores(j1, j2): copia el valor
 *   de tots els atributs de j2 cap a j1.
 * - funció compararJugadores(j1, j2): retorna -1 en
 *   cas que la millor pivot sigui j1, i 1 en cas
 *   que la millor sigui j2.
 */
#define MAX_NOM 20+1
#define MAX_COGNOM 20+1
#define MAX_JUGADORES 3

typedef struct {
    char nom[MAX_NOM];
    char cognom[MAX_COGNOM];
    float rebots;
```

```
    float taps;
} tJugadora;

/* Predeclaracions de funcions/accions */
void llegirJugadora(tJugadora *j);
void mostrarJugadora(tJugadora j);
void copiarJugadora(tJugadora *desti, tJugadora origen);
int compararJugadores(tJugadora j1, tJugadora j2);

/* Programa principal */
int main(int argc, char **argv) {
    tJugadora vJugadores[MAX_JUGADORES];
    int i, resultat;

    /* Es crea la tJugadora fictícia
     * millorPivot que ens ajudarà a trobar
     * la millor opció d'entre totes les
     * candidates
     */
    tJugadora millorPivot;
    millorPivot.rebots = 0;
    millorPivot.taps = 0;

    /* Llegim totes les jugadores amb
     * l'acció llegirJugadora(). Aquesta
     * acció rep un paràmetre de sortida
     * (out), el qual contindrà la
     * jugadora llegit per teclat. Com que
     * es tracta d'un paràmetre de tipus
     * out, es realitzarà un pas per
     * referència (= passarem un punter)
     */
    for (i=0; i<MAX_JUGADORES; i++) {
        tJugadora jugadora;
        llegirJugadora(&jugadora);
        vJugadores[i] = jugadora;
    }

    /* Mostrem per pantalla quina
     * és la millor jugadora amb perfil
     * pivot defensiu. La idea és anar
     * recorrent una a una les jugadores
     * del vector i comparar-les amb millorPivot:
     * 1. Si la jugadora del vector és millor
     *    que millorPivot, copiarem les dades
```

```
     *    de la jugadora cap a millorPivot.
     * 2. Si millorPivot és millor que la
     *    jugadora del vector, no farem res.
     * En finalitzar el recorregut de totes
     * les jugadores del vector, tindrem que
     * millorPivot contindrà la jugadora
     * que estem buscant.
     */
    for (i=0; i<MAX_JUGADORES; i++) {
        resultat = compararJugadores(millorPivot, vJugadores[i]);
        if (resultat == 1) {
            copiarJugadora(&millorPivot, vJugadores[i]);
        }
    }

    printf("\nMillor opció com a pivot defensiu : ");
    mostrarJugadora(millorPivot);
    return 0;
}

/* Implementació de les funcions/accions */
void llegirJugadora(tJugadora *j) {
    printf("Introdueix les dades de la nova jugadora: \n");
    printf("\tNom: ");
    scanf("%s", j->nom);
    printf("\tCognom: ");
    scanf("%s", j->cognom);
    printf("\t>> Promigs per partit:\n");
    printf("\tRebots: ");
    scanf("%f", &j->rebots);
    printf("\tTaps: ");
    scanf("%f", &j->taps);
}

void mostrarJugadora(tJugadora j) {
    printf("\n%s, %s: %.1f rebots, %.1f taps \n", j.cognom, j.nom, j.rebots, j.taps);
}

void copiarJugadora(tJugadora *desti, tJugadora origen) {
    /* Recordem:
     * - si el paràmetre és un punter, l'accessor
     *   d'atributs serà '->'
     * - si el paràmetre és un valor, l'accessor
     *   d'atributs serà '.'
     */
```

```
    strcpy(desti->nom, origen.nom);
    strcpy(desti->cognom, origen.cognom);
    desti->rebots = origen.rebots;
    desti->taps = origen.taps;
}

int compararJugadores(tJugadora j1, tJugadora j2) {
    /* Estem buscant una jugadora que
     * tingui un perfil de pivot defensiu,
     * amb el que agafarem:
     * 1. Aquella que tingui més rebots per partit
     * 2. En cas d'empat de rebots, aquella que
     *    faci més taps per partit
     */
    if (j1.rebots > j2.rebots) {
        return -1;
    } else {
        if (j1.rebots < j2.rebots) {
            return 1;
        } else {
            /* En aquest punt tenim que
             * j1.rebots == j2.rebots,
             * amb el que anem a comparar el següent
             * atribut segons la prioritat definida
             * del perfil pivot defensiu
             */
            if (j1.taps > j2.taps) {
                return -1;
            } else {
                if (j1.taps < j2.taps) {
                    return 1;
                } else {
                    return 0;
                }
            }
        }
    }
}
```

Exemple d'execució:

```
Introdueix les dades de la nova jugadora:
    Nom: Julia
    Cognom: Sanz
    >> Promigs per partit:
    Rebots: 7.9
    Taps: 0.9
Introdueix les dades de la nova jugadora:
    Nom: Nuria
    Cognom: Gutierrez
    >> Promigs per partit:
    Rebots: 7.9
    Taps: 1.4
Introdueix les dades de la nova jugadora:
    Nom: Mireia
    Cognom: Mateu
    >> Promigs per partit:
    Rebots: 6.8
    Taps: 2.1

Millor opció com a pivot defensiu :
Gutierrez, Nuria: 7.9 rebots, 1.4 taps
```

6.5.1 Explicació sobre l'acció copiarJugadora()

L'acció `copiarJugadora(tJugadora *desti, tJugadora origen)` de l'exemple rep dos paràmetres:

- El primer d'ells és `desti`, un punter a una tupla de tipus `tJugadora`; una altra forma de dir-ho és que el valor de `desti` el **passem per referència**. Aquest paràmetre és de tipus `out`, ja que no utilitzem per res el valor inicial que té i només ens interessa el valor final que tindrà en executar l'acció.
- El segon és `origen`, una tupla que **passem per valor**. Per tant en aquest cas no estem passant el punter a una `tJugadora`, sinó directament una `tJugadora`.

Quan tenim un punter a una tupla, accedim a l'element mitjançant l'operador `->`. En canvi, si estem tractant una tupla accedirem a un atribut seu amb l'operador `.` (punt).

La codificació de l'acció és:

```c
void copiarJugadora(tJugadora *desti, tJugadora origen) {
    /* Recordem:
     * - si el paràmetre és un punter, l'accessor
     *   d'atributs serà '->'
     * - si el paràmetre és un valor, l'accessor
     *   d'atributs serà '.'
     */
    strcpy(desti->nom, origen.nom);
    strcpy(desti->cognom, origen.cognom);
    desti->rebots = origen.rebots;
    desti->taps = origen.taps;
}
```

No utilitzarem el prefix `&` davant de la `tJugadora origen`, de la mateixa forma que no utilitzem l'`&` quan anem a imprimir per pantalla amb `printf()` el valor d'una variable: com que estem tractant amb un valor, simplement hi accedim a ell i l'utilitzem. Per tant aquestes accions serien **incorrectes**:

```c
    strcpy(desti->nom, &origen.nom);
    strcpy(desti->cognom, &origen.cognom);
    desti->rebots = &origen.rebots;
    desti->taps = &origen.taps;
```

Sí que tenim una alternativa possible a l'operador `->`, segons s'indica a la xWiki:

```c
    strcpy((*desti).nom, origen.nom);
    strcpy((*desti).cognom, origen.cognom);
    (*desti).rebots = origen.rebots;
    (*desti).taps = origen.taps;
```

Per tant aquest darrer bloc de codi es pot substituir a l'exemple i tot continua funcionant correctament, ja que són equivalents. Es pot utilitzar una forma o l'altra, tot i que l'operador `->` sembla més fàcil d'entendre visualment.

6.6 Frequently Made Mistakes

6.6.1 Definició d'accions/funcions: noms de paràmetres

Pseudocodi incorrecte:

```
action hotelCmp(in tHotel, in tHotel)
    { ... }
end action
```

En l'exemple mostrat, sembla que la intenció és definir dos paràmetres, però falta indicar l'identificador (nom) per a cada un d'ells. Recordem que en la declaració de la capçalera d'una acció/funció, cal indicar, per a cada paràmetre:

- El tipus de paràmetre que rep l'acció o funció, (in, out o inout)
- El tipus de dades del paràmetre (tHotel en l'exemple)
- L'identificador associat al paràmetre (no definit en l'exemple).

Pseudocodi correcte:

```
action hotelCmp(in hotel1: tHotel, in hotel2: tHotel)
    { ... }
end action
```

6.6.2 Definició d'accions/funcions: tipus de paràmetres

Pseudocodi incorrecte:

```
action hotelCmp(in hotel1: tHotel, out hotel2: tHotel)
    hotel1.id = hotel2.id;
end action
```

En la **declaració de la capçalera** d'una acció/funció cal indicar per a cada paràmetre el tipus de paràmetre que rep l'acció o funció: `in`, `out` o `inout`. Els paràmetres de tipus `in` no es poden modificar a l'interior de l'acció, mentre sí que podem fer-ho en el cas dels paràmetres de tipus `out` o `inout`. En l'exemple, s'està modificant el paràmetre `hotel1` actualitzant el valor d'un dels seus camps.

O bé es tracta d'una confusió entre els dos paràmetres, o bé `hotel1` ha de ser de tipus `out`.

Pseudocodis correctes:

```
action hotelCmp(in hotel1: tHotel, out hotel2: tHotel)
    hotel2.id = hotel1.id;
end action
```

```
action hotelCmp(out hotel1: tHotel, in hotel2: tHotel)
    hotel1.id = hotel2.id;
end action
```

```
action hotelCmp(out hotel1: tHotel, inout hotel2: tHotel)
    hotel1.id = hotel2.id;
end action
```

Fixeu-vos que si el paràmetre `hotel1` és de tipus `out` i hem de llegir el paràmetre `hotel2`, aquest segon paràmetre ha de ser forçosament de tipus `in` o `inout`, per la qual cosa n'hem modificat el tipus a la capçalera de la funció.

6.6.3 Definició de funcions: tipus de dades de retorn

Pseudocodi incorrecte:

```
function hotelCmp(h1: tHotel, h2: tHotel)
    { ... }
end function
```

En l'exemple mostrat, `hotelCmp` està definida com a funció, amb els paràmetres d'entrada `h1` i `h2`, però les funcions sempre retornen un valor, i cal indicar-ne sempre el tipus de dades a la mateixa capçalera.

Pseudocodi correcte:

```
function hotelCmp(h1: tHotel, h2: tHotel): integer
    { ... }
end function
```

Observeu que en aquest cas, no cal indicar el tipus de paràmetres d'entrada (`in`, `out` o `inout`), perquè en les funcions els paràmetres sempre són d'entrada i no es poden modificar.

6.6.4 Definició d'accions: retorn de valor

Pseudocodi incorrecte:

```
action hoteTableInit(inout tHotelTable: tabHotels): void
    { ... }
end action
```

Aquí s'ha definit una acció anomenada `hotelTableInit`, que rep un paràmetre d'entrada/sortida (`inout`) de tipus `tHotelTable`, amb l'identificador `tabHotels`. Les accions mai retornen un valor, però erròniament s'ha interpretat que cal indicar aquest fet afegint `:void` al final de la capçalera de l'acció.

Aquest cas és doblement incorrecte degut a que `:void` és un identificador propi de C que no existeix en llenguatge algorísmic.

Pseudocodi correcte:

```
action hoteTableInit(inout tHotelTable: tabHotels)
    { ... }
end action
```

Observeu que en aquest cas, en tractar-se d'una **acció** cal indicar el tipus de paràmetres d'entrada: `in`, `out` o `inout`.

6.6.5 Definició d'accions/funcions: accions obertes

Pseudocodi incorrecte:

```
action hotelCmp(in hotel1: tHotel, out hotel2: tHotel)
    hotel1.id = hotel2.id;
    { ... }
action readHotel(out hotel1: tHotel)
    hotel1.id = readInteger();
end action
```

Les **funcions** i **accions** constitueixen **blocs de codi** amb unes funcionalitats i característiques especials, i com a tal han d'estar ben delimitades, mitjançant les paraules reservades `function` … `end function` i `action` … `end action`. Deixar una acció sense tancar pot semblar un tema menor, però en C això comportarà amb tota seguretat un error de compilació (sovint difícil de detectar).

Pseudocodi correcte:

```
action hotelCmp(in hotel1: tHotel, out hotel2: tHotel)
    hotel1.id = hotel2.id;
    { ... }
end action
action readHotel(out hotel1: tHotel)
    hotel1.id = readInteger();
end action
```

6.6.6 Crida a funcions/accions: tipus de paràmetres

Pseudocodi incorrecte:

```
hotelRead(out h1: tHotel);
hotelRead(out h2: tHotel);

action hotelRead(out h: tHotel)
    { ... }
end action
```

El tipus de paràmetre que rep una acció/funció, així com el tipus de dades i l'identificador del paràmetre, cal indicar-los únicament en la **definició de la**

capçalera de l'**acció**. En el cas de **funcions**, no indicarem el tipus de dades perquè sempre són de tipus `in`.

En la **crida** a una acció/funció únicament cal indicar els paràmetres.

Pseudocodi correcte:

```
hotelRead(h1);
hotelRead(h2);

action hotelRead(out h: tHotel)
    { ... }
end action
```

6.6.7 Crida a accions/funcions: paraules reservades

Pseudocodi incorrecte:

```
action hotelRead(h1);
action hotelRead(h2);

action hotelRead(out h: tHotel)
    { ... }
end action
```

La paraula reservada `action` cal indicar-la únicament en la **definició de la capçalera** de l'acció.

En la **crida** a una acció únicament cal indicar el nom de l'acció i els paràmetres; es incorrecto afegir novament la paraula reservada `action`.

Pseudocodi correcte:

```
hotelRead(h1);
hotelRead(h2);

action hotelRead(out h: tHotel)
    { ... }
end action
```

6.6.8 Crida a funcions: crida buida

Pseudocodi incorrecte:

```
bestHotel: integer;

compareHotels(h1,h2);

if (bestHotel = 1) then
    { ... }
end if

function hotelRead(h1: tHotel, h2: tHotel): integer
    { ... }
    return bestHotel;
end function
```

Com passa en totes les funcions, `hotelRead()` retorna un valor: en aquest cas, la variable `bestHotel`. No obstant, la **crida** que es fa de la funció no és correcta, perquè el retorn de la funció no es guarda enlloc, i la variable `bestHotel` del programa principal segueix sense haver-se actualitzat.

En el cas de C aquest codi no provocaria un error, però si no fem res més des del programa principal no tindrem accés al resultat que retorna la funció (que és el que se suposa que volíem fer).

Pseudocodi correcte:

```
bestHotel: integer;

bestHotel:= compareHotels(h1,h2);

if (bestHotel = 1) then
    {...}
end if

function hotelRead(h1: tHotel, h2: tHotel): integer
    {...}
    return bestHotel;
end function
```

El retorn d'una funció també es pot utilitzar en una expressió, en aquest cas

sense assignar-lo a cap variable, ja que en la mateixa expressió, el compilador calcularà el retorn de la funció i el substituirà el valor corresponent dins la funció:

Pseudocodi correcte:

```
bestHotel: integer;

if (compareHotels(h1,h2) = 1) then
    { ... }
end if

function hotelRead(h1: tHotel, h2: tHotel): integer
    { ... }
    return bestHotel;
end function
```

6.6.9 Crida a funcions: absència de paràmetres

Pseudocodi incorrecte:

```
var
    myReal: real;
    myInteger: integer;
end var

myInteger:= integerToReal();
```

La funció `integerToReal` retorna un valor i el guardem a la variable `myInteger`. El problema és que la funció necessita que li passem un **paràmetre** quan la cridem: en aquest cas, el nombre real a convertir a enter.

En cas de dubte, cal que consultem sempre la **capçalera de declaració** de la funció, ja que ens indicarà en cada cas el nombre i tipus de paràmetres que espera.

Pseudocodi correcte:

```
var
    myReal: real;
    myInteger: integer;
end var

myInteger:= integerToReal(myReal);
```

6.6.10 Crida a funcions: errors de sintaxi múltiples

Pseudocodi incorrecte:

```
var
    hotel1: tHotel;
end var

hotelRead(in: &myHotel);

action hotelRead(in myHotel: tHotel)
    { ... }
end action
```

El pseudocodi anterior mostra una barreja d'errors de sintaxi diversos en la crida a la funció (n'és només un exemple): - S'indica el tipus `in`. - S'inclou l'operador `:`. - S'afegeix sintaxi pròpia de C amb l'operador `&`, per passar el paràmetre `myHotel` per referència. En llenguatge algorísmic el pas d'un paràmetre per referència es determina mitjançant la capçalera de la funció.

Pseudocodi correcte:

```
var
    hotel1: tHotel;
end var

hotelRead(myHotel);

action hotelRead(in myHotel: tHotel)
    { ... }
end action
```

6.6.11 Definició de funcions: funcions dins el main

Codi incorrecte:

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

#define MAX_LEN 15

int main(int argc, char **argv) {
    int num1;
    int num2;
    bool myBool;

    bool compareNumbers(int n1, int n2){
       return (n1 > n2);
    }

    myBool = compareNumbers(num1, num2);
    return 0;
}
```

Les funcions i accions cal definir-les de forma individual i separada. No és correcte definir funcions niades o incloses dins d'altres (com per exemple el `main`), entre altres coses perquè aquestes funcions no estaran disponibles des de l'exterior de la funció on són creades.

Codi correcte:

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

#define MAX_LEN 15

int main(int argc, char **argv) {
    int num1;
    int num2;
    bool myBool;
    myBool = compareNumbers(num1, num2);
    return 0;
}
```

```
bool compareNumbers(int n1, int n2){
    return (n1 > n2);
}
```

Chapter 7

PAC07

7.1 Quan utilitzar & dins de funcions/accions

De vegades quan fem crides a **accions** dins d'**accions**, veiem que si passem un paràmetre amb `&` els resultats no són correctes, però en canvi passant el paràmetre sense l'`&` tot funciona. I a l'inrevés.

Per tal d'aclarir aquests dubtes utilitzarem el següent exemple: és un programa senzill que treballa amb tipus `tPac`. Un element `tPac` conté dos atributs: un `nom` (cadena de caràcters) i una `nota` (decimal). El programa realitza la lectura de valors des de teclat amb l'acció `pacRead(...)`, la modificació del nom de la PAC amb `nomToUpperCase(...)`, i la seva posterior impressió per pantalla mitjançant `pacWrite(...)`.

Amb l'objectiu de deixar-ho tot el més clar possible s'han afegit comentaris extensos dins del programa, explicant en cada situació què es fa i per quin motiu.

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

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

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

```
} tPac;

/* Predeclaració de funcions i accions */

/* Acció que llegeix per teclat els atributs d'un
 * tipus tPac i li assigna els valors. En aquest
 * cas el paràmetre pac és de tipus 'out', donat que
 * el seu valor abans i després d'executar l'acció
 * haurà canviat. A més, com que el valor inicial
 * de pac no ens interessa per res (recordem que
 * l'objectiu d'aquesta acció és llegir de teclat
 * i donar valor a pac, per tant sobreescriurem
 * qualsevol valor previ que tingui), podem
 * descartar que sigui de tipus 'inout'.
 * Quan un paràmetre és de tipus out/inout, el
 * passem per referència o, el que és el mateix,
 * passem un punter a un tipus d'element; com
 * es pot veure, aquí pac és un punter a un
 * element de tipus tPac, ja que va precedit per *.
 * Utilitzem un punter perquè és l'única manera
 * que tenim de modificar un element definit
 * fora de l'àmbit de l'acció, des de dins de
 * la pròpia acció.
 */
void pacRead(tPac *pac);

/* Acció que, donat un tipus tPac, mostra el seu
 * contingut (nom i nota) per pantalla. Aquí el
 * paràmetre pac és de tipus 'in': el seu valor
 * abans i després d'executar l'acció no variarà.
 * Un paràmetre de tipus 'in' el passem per valor:
 * en aquest cas és un element de tipus tPac.
 * Com es pot veure, no ha d'anar precedit per *.
 */
void pacWrite(tPac pac);

/* Acció que, donat un tipus tPac, agafa el seu
 * nom i el passa a majúscules. Per exemple, si
 * el nom és "pAc01" el modificarà a "PAC01".
 * El paràmetre és de tipus 'inout': el seu valor
 * abans i després d'executar l'acció haurà canviat
 * i a més a més, el valor inicial que té és
 * important, ja que el necessitem per calcular el
 * valor final ("pAc01" --> "PAC01"). En ser un
 * paràmetre de tipus 'inout', passarem el seu
```

```
 * valor per referència: necessitem que pugui ser
 * modificat des de dins de l'acció, amb el que
 * és necessari treballar amb un punter a l'element
 * pac, d'aquí que vagi precedit amb *.
 */
void nomToUpperCase(tPac *pac);

/* Programa principal */
int main(int argc, char **argv) {
    /* Definim les variables */
    tPac pac1, pac2, pac3;

    /* Donem valor als atributs de cadascuna
     * de les 3 PAC. Fixeu-vos que estem passant
     * punters a elements de tipus tPac: el &
     * previ indica que agafem la direcció de
     * memòria on resideix l'element de tipus
     * tPac. Com que passem punters a memòria,
     * des de dins de l'acció podrem modificar
     * el contingut de pac1, pac2 i pac3, tot
     * i que aquestes tres variables han estat
     * definides fora de l'àmbit de l'acció.
     */
    pacRead(&pac1);
    pacRead(&pac2);
    pacRead(&pac3);

    /* Mostrem per pantalla els atributs de
     * cadascuna de les 3 PAC. En aquest cas
     * el pas de paràmetres es fa per valor:
     * passem directament els elements de tipus
     * tPac, ja que aquests no seran modificats
     * des de dins de l'acció (són de tipus 'in').
     */
    pacWrite(pac1);
    pacWrite(pac2);
    pacWrite(pac3);
    return 0;
}

/* Implementació de funcions i accions */

void pacRead(tPac *pac) {
    /* Llegim des de teclat el valor
     * corresponent al nom de la PAC
```

```
     */
    printf("Introdueix nom : ");
    scanf("%s", pac->nom);

    /* Llegim des de teclat el valor
     * corresponent a la nota de la PAC
     */
    printf("Introdueix nota: ");
    scanf("%f", &pac->nota);

    /* Ara arribem en un punt on, dins d'una
     * acció, cridem a un altra acció. Hem de
     * passar com a atribut pac? o bé &pac?.
     * Davant d'aquest dubte, ens hem de
     * preguntar quin tipus de valor conté ara
     * mateix (abans d'executar la següent acció)
     * el paràmetre pac. Si recordem com està
     * definida l'acció pacRead(tPac *pac), pac
     * és un punter a memòria, ja que va precedit
     * per *. Això significa que en aquest precís
     * moment pac continua sent un punter.
     * D'altra banda, si ens fixem amb la definició
     * de l'acció nomToUpperCase(tPac *pac), veiem
     * que també espera rebre un punter. Així com
     * que pac és un punter i nomToUpperCase(...)
     * espera rebre un punter, simplement li passem
     * pac com a paràmetre (i no pas &pac!)
     */
    nomToUpperCase(pac);
    printf("----------------------\n");
}

void pacWrite(tPac pac) {
    /* Mostrem per pantalla els valors
     * dels atributs de la PAC
     */
    printf(">> %s amb nota %.1f \n", pac.nom, pac.nota);
}

void nomToUpperCase(tPac *pac) {
    /* Hi ha llibreries que ja implementen els canvis
     * a majúscules o minúscules. En aquest cas però
     * hem optat per no utilitzar-ne cap, i implementar-la
     * nosaltres mateixos, tractant l'string nom de pac
     * com a un recorregut caràcter a caràcter.
```

```
     */
    int i = 0;
    for (i = 0; pac->nom[i] != '\0'; i++) {
        if (pac->nom[i] >= 'a' && pac->nom[i] <= 'z') {
            pac->nom[i] = pac->nom[i] + ('A' - 'a');
        }
    }
}
```

7.2 Exemple: com modular un programa amb CodeLite

A continuació explicaré detalladament com modular el programa típic `HelloWorld` d'una forma diferent de com s'explica als vídeos de la xWiki, que pot ser més entenedora. L'explicació és una mica extensa per tal que quedi el més clar possible.

7.2.1 Funcionament per defecte d'un projecte a CodeLite:

Quan a CodeLite creem un nou projecte, per exemple `Modularitat`, amb el típic `main.c` inicial que conté el programa `HelloWorld`, tenim la següent distribució d'arxius

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l
total 12
-rw-r--r-- 1 uoc uoc   93 nov 17 23:56 main.c
-rw-rw-r-- 1 uoc uoc 4274 nov 17 23:56 Modularitat.project
```

Si compilem aquest programa a CodeLite, veiem que ens ha generat una carpeta nova anomenada `Debug`:

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l
total 24
drwxrwxr-x 2 uoc uoc 4096 nov 17 23:58 Debug
-rw-r--r-- 1 uoc uoc   93 nov 17 23:56 main.c
-rw-rw-r-- 1 uoc uoc 3249 nov 17 23:58 Modularitat.mk
-rw-rw-r-- 1 uoc uoc 4274 nov 17 23:56 Modularitat.project
-rw-rw-r-- 1 uoc uoc   17 nov 17 23:58 Modularitat.txt
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$
```

El contingut de la carpeta `Debug` és el programa executable final juntament amb els arxius objecte intermitjos:

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ cd Debug
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat/Debug$ ls -l
total 24
-rw-rw-r-- 1 uoc uoc 4620 nov 17 23:58 main.c.o
-rw-rw-r-- 1 uoc uoc   23 nov 17 23:58 main.c.o.d
-rwxrwxr-x 1 uoc uoc 9708 nov 17 23:58 Modularitat
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat/Debug$
```

Si es vol, des d'un terminal de Lubuntu es pot executar el programa resultant `Modularitat` de la següent forma:

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat/Debug$ ./Modularitat
hello world
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat/Debug$
```

7.2.2 Executable dins de /bin.

Anem ara a modificar CodeLite per tal que l'executable ens el guardi sempre dins de la carpeta `./bin`.

El primer video de la xWiki indica com canviar la carpeta on es desa l'arxiu executable resultant de compilar el nostre programa. L'objectiu és que en comptes d'utilitzar la carpeta `./Debug` l'arxiu executable es desi dins de la nova carpeta `./bin`

Atenció: aquest apartat no es demana explícitament a la PAC07 (únicament es parla de les carpetes `./src` i `./include`), amb el que no cal aplicar-ho a la PAC.

Per fer aquest canvi, fem clic botó dret del ratolí sobre el nom del projecte, **Modularitat** -> **Settings...** -> **General** -> **Output File**: hi posem el valor `./bin/$(ProjectName)`. Això significa que l'executable es desarà dins de la carpeta `./bin` i que el seu nom serà el mateix que el nom del projecte (en el cas que tractem, s'anomenarà `Modularitat`)

A més, sense sortir d'aquesta mateixa pantalla, canviem el valor del **Working Directory** per : `./bin`. Aquest valor és el que té en consideració CodeLite a l'hora de buscar l'executable del projecte, per quan fem el `Run`.

Guardem els canvis i si ara compilem novament el programa, veiem que ha creat la carpeta `./bin` dins del nostre projecte:

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l
total 28
drwxrwxr-x 2 uoc uoc 4096 nov 18 00:11 bin
drwxrwxr-x 2 uoc uoc 4096 nov 17 23:58 Debug
-rw-r--r-- 1 uoc uoc   93 nov 17 23:56 main.c
-rw-rw-r-- 1 uoc uoc 3230 nov 18 00:11 Modularitat.mk
```

```
-rw-rw-r-- 1 uoc uoc 4350 nov 18 00:10 Modularitat.project
-rw-rw-r-- 1 uoc uoc   17 nov 18 00:11 Modularitat.txt
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$
```

Igual que en el cas anterior, si volem podem executar el programa des del propi terminal:

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ cd ./bin
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat/bin$ ls -l
total 12
-rwxrwxr-x 1 uoc uoc 9708 nov 18 00:11 Modularitat
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat/bin$ ./Modularitat
hello world
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat/bin$
```

A partir d'aquest moment, sempre que compilem el nostre programa, guardarà l'executable dins de la carpeta `./bin`.

7.2.3 Modularització del programa en un arxiu de capçaleres i un de funcions.

Centrarem ara l'explicació amb l'estructura que es demana a la PAC07 i treballant sobre el mateix exemple `Modularitat`.

El que volem aconseguir és la següent estructura:

```
projecte Modularitat
   |
   |-- /include/
   |     |
   |     \-- helloWorld.h
   |
   \-- /src/
         |
         |-- helloWorld.c
         \-- main.c
```

L'objectiu és dividir (modularitzar) l'arxiu únic `main.c` que tenim fins ara en tres arxius:

- `helloWorld.h` : aquest arxiu contindrà la predeclaració de totes les funcions i accions del nostre programa, així com la definició de tots els tipus necessaris (enumeratius, tuples, etc) i constants que requereixi el nostre programa C. Aquest arxiu l'ubicarem dins de la carpeta `./include` del nostre projecte
- `helloWorld.c` : dins d'ell s'implementaran totes les accions i funcions del programa. Bàsicament contindrà el codi de tot allò que hem predeclarat

a l'arxiu `helloWorld.h`.
- `main.c` : contindrà el codi del programa principal, identificat per la funció `main()`.

El primer pas que podem fer és crear les dues carpetes que necessitarà el nostre projecte modularitzat, `src` i `include` :

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ mkdir src
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ mkdir include
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l
total 36
drwxrwxr-x 2 uoc uoc 4096 nov 18 00:14 bin
drwxrwxr-x 2 uoc uoc 4096 nov 18 00:14 Debug
drwxrwxr-x 2 uoc uoc 4096 nov 18 00:18 include
-rw-r--r-- 1 uoc uoc   93 nov 17 23:56 main.c
-rw-rw-r-- 1 uoc uoc 3230 nov 18 00:14 Modularitat.mk
-rw-rw-r-- 1 uoc uoc 4350 nov 18 00:10 Modularitat.project
-rw-rw-r-- 1 uoc uoc   17 nov 18 00:14 Modularitat.txt
drwxrwxr-x 2 uoc uoc 4096 nov 18 00:18 src
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$
```

El següent pas és transformar el programa `HelloWorld` típic en un programa modularitzat: per aquest motiu ens cal la definició d'algunes funcions o accions que ens permetin separar un únic arxiu en un de capçaleres (`helloWorld.h`), un d'implementació de funcions/accions (`helloWorld.c`) i un principal (`main.c`). Per tant, l'objectiu és passar del següent programa… :

```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("hello world\n");
    return 0;
}
```

… al següent codi modularitzat:

```c
#include <stdio.h>

/* Predeclaració de les funcions/accions */
void showHelloMessage();
```

```c
/* Codi principal */
int main(int argc, char **argv) {
    showHelloMessage();
    return 0;
}

/* Implementació de les funcions/accions */
void showHelloMessage() {
    printf("hello world\n");
};
```

Si s'executa el nou codi dins d'un únic `main.c`, aquest seguirà mostrant correctament el missatge de *"hello world"*. L'objectiu és acabant-lo dividint en tres blocs:

```c
#include <stdio.h>

/* Inici contingut de l'arxiu helloWorld.h */
/* Predeclaració de les funcions/accions */
void showHelloMessage();
/* Fi contingut de l'arxiu helloWorld.h */

/* Inici contingut de l'arxiu main.c */
/* Codi principal */
int main(int argc, char **argv) {
    showHelloMessage();
    return 0;
}
/* Fi contingut de l'arxiu main.c */

/* Inici contingut de l'arxiu helloWorld.c */
/* Implementació de les funcions/accions */
void showHelloMessage() {
    printf("hello world\n");
};
/* Fi contingut de l'arxiu helloWorld.c */
```

El primer pas que farem serà eliminar des de CodeLite el programa vell; per tant, **CodeLite** -> projecte **Modularitat** -> **src** -> botó dret sobre el nom de l'arxiu `main.c` -> **Remove** -> confirmem l'esborrat -> confirmem l'esborrat de l'arxiu `main.c` del disc.

En aquest punt, a CodeLite, dins del projecte `Modularitat` únicament hi tenim

una carpeta *virtual* anomenada `src`. Atenció: la carpeta *virtual* anomenada `src` del nostre projecte de CodeLite en aquests moments no té cap relació amb la carpeta `./src` que hem creat fa un moment.

Creem el programa principal main.c fent: **CodeLite** -> projecte **Modularitat** -> botó dret sobre carpeta **src** -> **Add New File** -> seleccionem el tipus **C Source File (.c)** -> indiquem com a **Name**: `main.c`, i com a **Location** seleccionem la carpeta `./src` que hem creat abans.

Anem a crear el segon arxiu, `helloWorld.c`, exactament de la mateixa forma: **CodeLite** -> projecte **Modularitat** -> botó dret sobre carpeta **src** -> **Add New File** -> seleccionem el tipus **C Source File (.c)** -> indiquem com a **Name**: `helloWorld.c`, i com a **Location** seleccionem la carpeta `./src` creada anteriorment.

Creem ara una carpeta *virtual* dins del nostre projecte amb **CodeLite** -> botó dret sobre el projecte **Modularitat** -> **New Virtual Folder** -> li posem per nom: `include`

Per finalitzar, creem el tercer arxiu requerit: `helloWorld.h`. Els passos seran: **CodeLite** -> projecte **Modularitat** -> botó dret sobre carpeta **include** -> **Add New File** -> seleccionem el tipus **Header File (.h)** -> indiquem com a **Name**: `helloWorld.h`, i com a **Location** seleccionem la carpeta `./include` creada anteriorment.

En aquest punt el nostre projecte `Modularitat` a CodeLite tindrà l'estructura desitjada:

```
projecte Modularitat
   |
   |-- /include/
   |     |
   |     \-- helloWorld.h
   |
   \-- /src/
         |
         |-- helloWorld.c
         \-- main.c
```

Ara només cal donar contingut als 3 arxius buits que hem creat.

Comencem per l'arxiu de capçaleres `helloWorld.h`. L'editem com ho fem habitualment a CodeLite, i li copiem el fragment de codi que hem dit abans que li corresponia:

```c
#include <stdio.h>

/* Predeclaració de les funcions/accions */
void showHelloMessage();
```

Editem l'arxiu `helloWorld.c` i li afegim el codi corresponent:

```c
/* Implementació de les funcions/accions */
void showHelloMessage() {
    printf("hello world\n");
};
```

I per finalitzar editem el contingut de l'arxiu `main.c` amb el codi comentat anteriorment:

```c
/* Codi principal */
int main(int argc, char **argv) {
    showHelloMessage();
    return 0;
}
```

En aquest punt tenim el programa modularitzat, però ens falta dues qüestions:

- Configurar CodeLite per tal que tingui presents on buscar l'arxiu de capçaleres `.h` quan hi fem referència.
- Fer que els arxius estiguin enllaçats entre ells: ara mateix són completament independents.

Anem pel primer punt: per indicar a CodeLite on trobarà tots els arxius `.h` d'un programa, només cal fer **CodeLite** -> clic botó dret sobre **Modularitat** -> **Settings...** -> **Compiler** -> dins de l'opció **Include Paths** afegir el valor `.;./include`

Pel segon punt: tenim que l'arxiu `main.c` en aquests moments fa referència a una acció anomenada `showHelloMessage()`, de la qual no en sabem res. El que ens cal és importar l'arxiu de capçaleres; com que és un arxiu que hem creat nosaltres, l'`include` va entre cometes dobles:

```c
#include "helloWorld.h"

/* Codi principal */
int main(int argc, char **argv) {
    showHelloMessage();
    return 0;
}
```

El següent pas serà afegir l'`include` habitual de la llibreria `stdio.h` dins de l'arxiu `helloWorld.c`, ja que en ell utilitzem la funció `printf()` inclosa en aquesta llibreria:

```c
#include <stdio.h>

/* Implementació de les funcions/accions */
void showHelloMessage() {
    printf("hello world\n");
};
```

En aquest punt ja podem executar el nostre programa modularitzat correctament.

L'estructura resultant i el contingut de cada carpeta és el desitjat:

```
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l ./bin
total 12
-rwxrwxr-x 1 uoc uoc 10172 nov 18 01:09 Modularitat
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l ./include/
total 4
-rw-rw-r-- 1 uoc uoc 70 nov 18 00:54 helloWorld.h
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l ./src
total 8
-rw-rw-r-- 1 uoc uoc 122 nov 18 01:09 helloWorld.c
-rw-rw-r-- 1 uoc uoc 113 nov 18 01:07 main.c
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$ ls -l ./Debug/
total 20
-rw-rw-r-- 1 uoc uoc 4544 nov 18 01:09 src_helloWorld.c.o
-rw-rw-r-- 1 uoc uoc   43 nov 18 01:09 src_helloWorld.c.o.d
-rw-rw-r-- 1 uoc uoc 2440 nov 18 01:09 src_main.c.o
-rw-rw-r-- 1 uoc uoc   75 nov 18 01:09 src_main.c.o.d
uoc@fp:~/Documents/codelite/workspaces/Test/Modularitat$
```

Ni per la realització d'aquest exemple ni per la PAC07 cal crear llibreries pròpies, amb el que aquest punt no el comento: únicament tractarem amb arxius de

capçalera `.h`, arxius d'implementació `.c`, i l'arxiu corresponent al codi principal `main.c`.

7.3 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
```

7.4 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 0xbfb86c4
```

La funció `scanf()` com a tal retorna un valor, tot i que nosaltres no l'utilitzem habitualment: 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, imaginem 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:

```
#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 variï (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:

```c
#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.

7.5 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).

7.6 Exemple: capgirarParaula

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 totes dues volem aplicar l'acció `capgirar()`. Una possible forma d'implementar-ho tot plegat seria la següent:

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

#define MAX_CHAR 20+1
#define MAX_PARAULES 2

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, paraul
    }
}

/* 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`, ja que únicament s'executa sobre una `tParaula`. Com la volem executar per a cadascuna de les dues `tParaula`, repetim la crida dues vegades dins del bucle, una per cada`tParaula`.

7.7 Exemple: isParell

Exemple de funció que retorna un booleà:

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

/* Predeclaració de la funció isParell, la qual retorna un
 * booleà que indica si el número passat per paràmetre és
 * parell (true) o no (false).
 */
```

```c
bool 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 */
bool isParell(int numero) {
    if (numero % 2 == 0) {
        return true;
    } else {
        return false;
    }
}
```

7.8 Exemple: pivotDefensiuTirsLliures

S'afegeix a l'exemple de les jugadores de bàsquet un nou factor de comparació: en cas d'empat de les comparacions anteriors, escollirem la que tingui un millor percentatge de tirs lliures.

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

/* Rebem una petició d'un equip femení de bàsquet,
 * en el qual ens demanen un programa que els permeti
 * seleccionar la millor pivot defensiu d'entre una
 * sèrie de candidates.
 * La millor pivot defensiu és aquella que captura
 * més rebots; en cas d'empat, s'escollirà la que
```

```
 * faci més taps. En cas d'empat ens interessarà
 * escollir la que tingui millor percentatge
 * de tirs lliures.
 * Caldrà implementar 3 accions i 2 funcions:
 * - acció llegirJugadora(j): llegeix de teclat i
 *   guarda tots els atributs de la jugadora a la
 *   tupla j.
 * - acció mostrarJugadora(j): mostra per pantalla
 *   el valor dels atributs de la tupla j.
 * - acció copiarJugadores(j1, j2): copia el valor
 *   de tots els atributs de j2 cap a j1.
 * - funció compararJugadores(j1, j2): retorna -1 en
 *   cas que la millor pivot sigui j1, i 1 en cas
 *   que la millor sigui j2.
 * - funció percentatgeTirsLliures(intentats, encertats):
 *   retorna el percentatge d'encert en tirs lliures
 *   en funció dels valors passats per paràmetre.
 */

#define MAX_NOM 20+1
#define MAX_COGNOM 20+1
#define MAX_JUGADORES 3

typedef struct {
    char nom[MAX_NOM];
    char cognom[MAX_COGNOM];
    float rebots;
    float taps;
    float tirsLliures; /* en percentatge */
} tJugadora;

/* Predeclaracions */
void llegirJugadora(tJugadora *j);
void mostrarJugadora(tJugadora j);
void copiarJugadora(tJugadora *desti, tJugadora origen);
int compararJugadores(tJugadora j1, tJugadora j2);
float percentatgeTirsLliures(int intentats, int encertats);

/* Programa principal */
int main(int argc, char **argv) {
    tJugadora vJugadores[MAX_JUGADORES];
    int i, resultat;

    /* Es crea la tJugadora fictícia
     * millorPivot que ens ajudarà a trobar
```

```
     * la millor opció d'entre totes les
     * candidates
     */
    tJugadora millorPivot;
    millorPivot.rebots = 0;
    millorPivot.taps = 0;
    millorPivot.tirsLliures = 0.0;

    /* Llegim totes les jugadores amb
     * l'acció llegirJugadora(). Aquesta
     * acció rep un paràmetre de sortida
     * (out), el qual contindrà la
     * jugadora llegit per teclat. Com que
     * es tracta d'un paràmetre de tipus
     * out, es realitzarà un pas per
     * referència (= passarem un punter)
     */
    for (i=0; i<MAX_JUGADORES; i++) {
        tJugadora jugadora;
        llegirJugadora(&jugadora);
        vJugadores[i] = jugadora;
    }

    /* Mostrem per pantalla quina
     * és la millor jugadora amb perfil
     * pivot defensiu. La idea és anar
     * recorrent una a una les jugadores
     * del vector i comparar-les amb millorPivot:
     * 1. Si la jugadora del vector és millor
     *    que millorPivot, copiarem les dades
     *    de la jugadora cap a millorPivot.
     * 2. Si millorPivot és millor que la
     *    jugadora del vector, no farem res.
     * En finalitzar el recorregut de totes
     * les jugadores del vector, tindrem que
     * millorPivot contindrà la jugadora
     * que estem buscant.
     * */
    for (i=0; i<MAX_JUGADORES; i++) {
        resultat = compararJugadores(millorPivot, vJugadores[i]);
        if (resultat != -1) {
            copiarJugadora(&millorPivot, vJugadores[i]);
        }
    }
```

```
    printf("\nMillor opció com a pivot defensiu : ");
    mostrarJugadora(millorPivot);
    return 0;
}

/* Implementació de les accions */
void llegirJugadora(tJugadora *j) {
    int intentats;
    int encertats;
    printf("Introdueix les dades de la nova jugadora: \n");
    printf("\tNom: ");
    scanf("%s", j->nom);
    printf("\tCognom: ");
    scanf("%s", j->cognom);
    printf("\t>> Promigs per partit:\n");
    printf("\tRebots: ");
    scanf("%f", &j->rebots);
    printf("\tTaps: ");
    scanf("%f", &j->taps);
    printf("\tTirs lliures intentats: ");
    scanf("%d", &intentats);
    printf("\tTirs lliures encertats: ");
    scanf("%d", &encertats);
    j->tirsLliures = percentatgeTirsLliures(intentats, encertats);
}

void mostrarJugadora(tJugadora j) {
    printf("\n%s, %s: %.1f rebots, %.1f taps, %.1f%% tirs lliures \n", j.cognom, j.nom
}

void copiarJugadora(tJugadora *desti, tJugadora origen) {
    /* Recordem:
     * - si el paràmetre és un punter, l'accessor
     *   d'atributs serà '->'
     * - si el paràmetre és un valor, l'accessor
     *   d'atributs serà '.'
     */
    strcpy(desti->nom, origen.nom);
    strcpy(desti->cognom, origen.cognom);
    desti->rebots = origen.rebots;
    desti->taps = origen.taps;
    desti->tirsLliures = origen.tirsLliures;
}

int compararJugadores(tJugadora j1, tJugadora j2) {
```

```
    /* Estem buscant una jugadora que
     * tingui un perfil de pivot defensiu,
     * amb el que agafarem:
     * 1. Aquella que tingui més rebots per partit
     * 2. En cas d'empat de rebots, aquella que
     *    faci més taps per partit
     * 3. En cas d'empat, la que tingui millor
     *    percentatge de tirs lliures
     */
    if (j1.rebots > j2.rebots) {
        return -1;
    } else {
        if (j1.rebots < j2.rebots) {
            return 1;
        } else {
            /* En aquest punt tenim que
             * j1.rebots == j2.rebots,
             * amb el que anem a comparar el següent
             * atribut segons la prioritat definida
             * del perfil pivot defensiu
             */
            if (j1.taps > j2.taps) {
                return -1;
            } else {
                if (j1.taps < j2.taps) {
                    return 1;
                } else {
                    /* Afegim la variant de valorar
                     * el percentatge de tirs lliures
                     */
                    if (j1.tirsLliures >= j2.tirsLliures) {
                        return -1;
                    } else {
                        return 1;
                    }
                }
            }
        }
    }
}

float percentatgeTirsLliures(int intentats, int encertats) {
    return ((float)encertats/intentats)*100.0;
}
```

Chapter 8

PAC08

8.1 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.

8.2 Exemple: calcularNota

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 una 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;

/* 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);
```

```c
/* 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 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).

8.3 Exemple: calcularNota amb introducció iterativa

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
```

```c
/* 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;
}
```

8.4 Exemple: recorregut vs cerca

El següent exemple es tracten els conceptes de **recorregut** i de **cerca** dels elements d'una taula. Dins del codi s'han afegit comentaris detallats per tal que cada pas quedi explicat.

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

/* Ens demanen un programa que permeti registrar
 * jugadores de bàsquet dins d'una taula. En una
 * mateixa taula tindrem tant les jugadores locals
 * com les visitants. Sobre elles hi realitzarem
 * un recorregut, consistent en mostrar-les totes
 * per pantalla, i també una cerca, on a partir
 * de l'equip i el dorsal mostrarem per pantalla
 * la jugadora en cas d'existir.
 * Ens caldrà implementar les següents accions:
 * - inicialitzarTaula(...): per inicialitzar la
 *   taula de jugadores.
 * - afegirJugadora(...) : per afegir una
 *   jugadora a una taula.
 * - mostrarJugadores(...) : per mostrar per
 *   pantalla totes les jugadores de la taula.
 * - cercarJugadora(...) : per buscar dins de la
 *   taula la jugadora que tingui un equip i
 *   un dorsal determinat.
 */

#define MAX_NOM 20+1
#define MAX_COGNOM 20+1
```

```c
#define MAX_JUGADORES 10

typedef enum {LOCAL, VISITANT} tEquip;

typedef struct {
    char nom[MAX_NOM];
    char cognom[MAX_COGNOM];
    int dorsal;
    tEquip equip;
} tJugadora;

/* Definició del tipus taula de
 * jugadores; com es pot veure, tal i
 * com s'indica a les xWiki, únicament
 * està formada per un vector i un
 * comptador de les jugadores que conté.
 */
typedef struct {
    tJugadora jugadores[MAX_JUGADORES];
    int nJugadores;
} tTaulaJugadores;

/* Predeclaracions */
void llegirJugadora(tJugadora *j);
void mostrarJugadora(tJugadora j);
void copiarJugadora(tJugadora *desti, tJugadora origen);
void inicialitzarTaula(tTaulaJugadores *taula);
void afegirJugadora(tTaulaJugadores *taula, tJugadora j);
void mostrarJugadores(tTaulaJugadores taula);
void cercarJugadora(tTaulaJugadores taula, int dorsal, tEquip equip);

/* Programa principal */
int main(int argc, char **argv) {
    tJugadora jugadora;
    tTaulaJugadores taula;
    tEquip equip;
    int dorsal, i;

    /* Per inicialitzar una taula simplement
     * posem el comptador d'elements a 0. Com
     * que la inserció de jugadores a la taula
     * té present aquest comptador, sobreescriurà
     * qualsevol altre valor que existís en
     * aquesta mateixa posició.
     */
```

```
    inicialitzarTaula(&taula);

    /* Demanem les jugadores per teclat i tot
     * seguit les anem afegint a la taula.
     */
    for (i=0; i<MAX_JUGADORES; i++) {
        llegirJugadora(&jugadora);
        afegirJugadora(&taula, jugadora);
    }

    /* Es recorren tots els elements
     * de la taula de jugadores per
     * tal d'imprimir les dades per
     * pantalla
     */
    mostrarJugadores(taula);

    printf("\nQuina jugadora vols cercar? : ");
    printf("\n>> Equip (0=LOCAL, 1=VISITANT) : ");
    scanf("%u", &equip);
    printf(">> Dorsal : ");
    scanf("%d", &dorsal);

    /* Es fa una cerca entre totes les
     * jugadores de la taula, de forma que
     * mostrarà per pantalla el resultat
     * obtingut; si no se'n troba cap,
     * retorna un missatge informatiu.
     */
    cercarJugadora(taula, dorsal, equip);
    return 0;
}

/* Implementació de les accions */
void llegirJugadora(tJugadora *j) {
    printf("Introdueix les dades de la nova jugadora: \n");
    printf("\tNom: ");
    scanf("%s", j->nom);
    printf("\tCognom: ");
    scanf("%s", j->cognom);
    printf("\tEquip (0=LOCAL, 1=VISITANT): ");
    scanf("%u", &j->equip);
    printf("\tDorsal: ");
    scanf("%d", &j->dorsal);
}
```

```c
void mostrarJugadora(tJugadora j) {
    if (j.equip == LOCAL) {
        printf("LOCAL   : %d %s,%s\n", j.dorsal, j.cognom, j.nom);
    } else {
        printf("VISITANT: %d %s,%s\n", j.dorsal, j.cognom, j.nom);
    }
}

void copiarJugadora(tJugadora *desti, tJugadora origen) {
    strcpy(desti->nom, origen.nom);
    strcpy(desti->cognom, origen.cognom);
    desti->dorsal = origen.dorsal;
    desti->equip = origen.equip;
}

void inicialitzarTaula(tTaulaJugadores *taula) {
    taula->nJugadores= 0;
}

void afegirJugadora(tTaulaJugadores *taula, tJugadora j) {

    /* Es comprova primer que la taula no estigui plena! */
    if(taula->nJugadores >= MAX_JUGADORES) {
        printf("Error en afegir jugadora a la taula\n");

    } else {
        /* S'afegeix la jugadora a la taula, i s'incrementa
           el comptador d'elements de la taula */
        copiarJugadora(&taula->jugadores[taula->nJugadores], j);

        /* Important!! després d'afegir una jugadora, no hem
         * d'oblidar incrementar el comptador! */
        taula->nJugadores++;
    }
}

void mostrarJugadores(tTaulaJugadores taula) {
    int i;
    i = 0;
    printf("\nRecorregut: jugadores entrades a la taula ...\n");
    while (i < taula.nJugadores) {
        mostrarJugadora(taula.jugadores[i]);
        i = i+1;
    }
}
```

```
void cercarJugadora(tTaulaJugadores taula, int dorsal, tEquip equip) {
    int i;
    bool trobada;

    /* El booleà 'trobada' s'utilitzarà
     * per sortir del bucle que recorre totes
     * les jugadores de la taula: quan en trobi
     * una, trobada = true, i la condició d'entrada
     * del bucle ja no es complirà, amb el que
     * la cerca haurà finalitzat.
     */
    trobada = false;

    if (equip == LOCAL) {
        printf("\nCerca: jugadora local amb dorsal núm. %d ...", dorsal);
    } else {
        printf("\nCerca: jugadora visitant amb dorsal núm. %d ...", dorsal);
    }

    for(i=0; i<taula.nJugadores && !trobada; i++) {
        if (taula.jugadores[i].dorsal == dorsal &&
            taula.jugadores[i].equip == equip) {
                trobada = true;
        }
    }

    /* Es mostra per pantalla els resultats
     * obtinguts
     */
    if (!trobada) {
        printf("\n>> No s'ha trobat cap jugadora. \n");
    } else {
        printf("\n>> Jugadora trobada : \n");
        mostrarJugadora(taula.jugadores[i-1]);
    }
}
```

Chapter 9

PAC09

9.1 Exemple: llistaCartes

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.

Una possible implementació seria:

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

/* 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);
bool end(tCartesList llista, int pos);
bool emptyList(tCartesList llista);
bool 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;
}

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

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

bool 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 10

PAC10

Chapter 11

PR1

11.1 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.

11.2 Exemple: tupla dins de tupla

De vegades pot costar veure com treballar amb atributs d'una tupla que a la seva vegada està dins d'una altra tupla, si s'hi accedeix per valor, per referència (punter), etc. Per aquest motiu, s'adjunta el següent exemple inventat, en el qual s'hi han afegit comentaris per tal que es vegi clarament com treballar amb atributs d'una tupla continguda dins d'una altra tupla:

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

/* Un parking d'un centre comercial ens ha
 * encarregat una app que faciliti els seus
 * clients a localitzar on ha aparcat el seu
 * vehicle.
 *
 * El parking ja disposa de tres tipus de
 * càmeres, independents entre elles:
 *
 * - d'entrada: càmera posicionada a l'entrada
 *   del pàrking que, a més de llegir la matrícula,
 *   permet saber el tipus de vehicle.
 *
 * - d'accés: càmeres que ens permeten
 *   saber si un vehicle ha pujat o ha baixat
 *   una planta.
 *
 * - de planta: càmeres que ens permeten saber
 *   la fila i eĺ número de plaça on s'ha
 *   aparcat un vehicle.
 *
 * L'identificador de vehicle és la seva
 * matrícula.
 *
 * A partir de la informació obtinguda pels
 * tres conjunts de càmeres, caldrà indicar
 * a l'usuari on ha aparcat el seu vehicle.
 */

#define MAX_MATRICULA 7+1

typedef enum {COTXE, MOTO, FURGONETA} tTipus;

/* El lloc on aparca un vehicle ve donat
 * per tres valors: la planta, la fila i el
 * número de la plaça.
 */
typedef struct {
    int planta;
    char fila;
    int numero;
} tAparcament;
```

```c
/* El tipus tVehicle conté la seva matrícula
 * (identificador únic), el tipus de vehicle
 * detectat en la primera càmera (COTXE, MOTO,
 * FURGONETA), i el lloc on ha aparcat (tAparcament)
 */
typedef struct {
    char matricula[MAX_MATRICULA];
    tTipus tipus;
    tAparcament aparcament;
} tVehicle;

/* Predeclaració d'accions */
void inicialitzar(tVehicle *vehicle, tTipus tipus, char *matricula);
void pujarPlanta(tVehicle *vehicle);
void baixarPlanta(tVehicle *vehicle);
void aparcar(tVehicle *vehicle, char fila, int numero);
void obtenirPosicio(tVehicle);

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

    tVehicle vehicle;

    /* Es llegeix la matrícula del vehicle amb la
     * càmera d'entrada del pàrking i s'assigna
     * a l'element de tipus tVehicle del nostre
     * pàrking
     */
    printf("\n>> Entrada al pàrking \n");
    inicialitzar(&vehicle, COTXE, "7472GZZ");

    printf("\n>> Baixar una planta\n");
    baixarPlanta(&vehicle);

    printf("\n>> Baixar una planta\n");
    baixarPlanta(&vehicle);

    printf("\n>> Pujar una planta\n");
    pujarPlanta(&vehicle);

    printf("\n>> Baixar una planta\n");
    baixarPlanta(&vehicle);

    printf("\n>> Baixar una planta\n");
    baixarPlanta(&vehicle);
```

```
    printf("\n>> Aparcar\n");
    aparcar(&vehicle, 'C', 23);

    printf("\nAnem a comprar ...");
    printf("\n... passen més de 3 hores ...");
    printf("\n... i ens oblidem d'on hem aparcat el vehicle !!\n");

    printf("\nSolució: consultem la posició del nostre vehicle");
    printf("\na l'app del pàrquing : \n");
    obtenirPosicio(vehicle);
    return 0;
}

/* Com que la posició ve donada pels atributs planta, fila i
 * numero de tAparcament, és molt important que inicialitzem
 * els valors (sobretot per la planta). La planta inicial del
 * pàrking és la 0, i la resta de plantes són soterrades.
 */
void inicialitzar(tVehicle *vehicle, tTipus tipus, char *matricula) {
    strcpy(vehicle->matricula, matricula);
    vehicle->tipus = tipus;
    /* L'accés a l'atribut aparcament (tupla) el fem amb '->'
     * ja que es tracta d'un punter, i accedim
     * als atributs de la tupla aparcament amb '.'
     */
    vehicle->aparcament.planta = 0;
    vehicle->aparcament.fila = '-';
    vehicle->aparcament.numero = 0;
}

/* Quan pugem una planta, incrementem en 1 l'atribut
 * planta de la tupla tAparcament que conté la tupla tVehicle
 */
void pujarPlanta(tVehicle *vehicle) {
    vehicle->aparcament.planta = vehicle->aparcament.planta + 1;
}

/* Quan baixem una planta, decrementem en 1 l'atribut
 * planta de la tupla tAparcament que conté la tupla tVehicle
 */
void baixarPlanta(tVehicle *vehicle) {
    vehicle->aparcament.planta = vehicle->aparcament.planta - 1;
}

/* Quan aparquem el vehicle, donem valor als atributs
```

```
 * fila i numero de la tupla tAparcament que conté la
 * tupla tVehicle
 */
void aparcar(tVehicle *vehicle, char fila, int numero) {
    vehicle->aparcament.fila = fila;
    vehicle->aparcament.numero = numero;
}

/* Mostrem per pantalla la posició del tVehicle */
void obtenirPosicio(tVehicle vehicle) {
    if (vehicle.tipus == COTXE) {
        printf("\nCotxe %s : ", vehicle.matricula);
    } else {
        if (vehicle.tipus == MOTO) {
            printf("\nMoto %s : ", vehicle.matricula);
        } else {
            printf("\nFurgoneta %s : ", vehicle.matricula);
        }
    }

    /* En aquest cas l'accés a l'atribut aparcament (tupla)
     * es fa amb '.' ja que s'ha passat per valor (no és un punter)
     * i accedim als atributs de la tupla aparcament amb '.'
     */
    printf("planta %d, ", vehicle.aparcament.planta);
    printf("fila %c, ", vehicle.aparcament.fila);
    printf("número %d \n", vehicle.aparcament.numero);
}
```

El resultat de l'execució d'aquest programa és:

```
>> Entrada al pàrking

>> Baixar una planta

>> Baixar una planta

>> Pujar una planta

>> Baixar una planta

>> Baixar una planta
```

```
>> Aparcar

Anem a comprar regals de reis ...
... passem més de 3 hores ...
... i ens oblidem d'on hem aparcat el vehicle !!

Solució: consultem la posició del nostre vehicle
a l'app del parking :

Cotxe 7472GZZ : planta -3, fila C, número 23
```

11.3 Desplaçament d'elements en un vector

Els desplaçaments en un vector es poden produir quan s'afegeix o s'elimina un element d'un vector.

11.3.1 Afegir un element

Volem afegir un element dins del vector de la taula en una posició que no és l'última. Per exemple, si volem que els elements que afegim a un vector vagin a la posició inicial, caldrà abans desplaçar la resta d'elements una posició a la dreta:

Contingut inicial del vector:

`[element2] [element5] [element1] [element7]`

Abans d'afegir el nou `element6`, caldrà desplaçar tots els elements una posició cap a la dreta, per tal de fer un buit a l'inici:

`[........] [element2] [element5] [element1] [element7]`

Ara ja es pot afegir l'element6 a la primera posició del vector:

`[element6] [element2] [element5] [element1] [element7]`

Important: com a darrer pas, incrementar en 1 l'atribut que conté la mida de la taula.

11.3.2 Esborrar un element

Quan volem eliminar un element d'un vector també es pot produir un desplaçament d'elements. Per exemple:

Contingut inicial del vector:

`[element2] [element5] [element1] [element7]`

Per eliminar l'`element5` simplement desplacem la resta d'elements que van a continuació una posició cap a l'esquerra:

`[element2] [element1] [element7]`

Important: com a darrer pas, decrementar en 1 l'atribut que conté la mida de la taula.

11.3.3 Exemple: diesSetmana

A continuació s'adjunta un exemple inventat per consolidar l'explicació anterior: la idea és que pugueu comparar la inserció d'elements a una taula com hem fet habitualment fins ara (afegits al final de tot), amb una altra acció que permet afegir de forma ordenada els elements a la taula. En aquesta segona inserció caldrà anar realitzant desplaçaments cap a la dreta dels elements del vector de la taula per tal que vagin quedant ordenats.

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

/* Definició de constants */
#define MAX_DIES 7
#define MAX_NOM 9+1

/* Definició de la tupla tDia */
typedef struct {
    char nom[MAX_NOM];
    int id;
} tDia;

/* Definició de la taula tSetmana */
typedef struct {
    tDia dies[MAX_DIES];
    int numDies;
} tSetmana;

/* Predeclaració de les accions */

/* Acció que inicialitza la taula tSetmana,
 * tot posant l'atribut numDies a 0
```

```
 */
void inicialitzar(tSetmana *setmana);

/* Acció que afegeix un element de tipus tDia
 * a la taula tSetmana
 */
void afegirDia(tSetmana *setmana, tDia dia);

/* Acció que afegeix un element de tipus tDia
 * a la taula tSetmana de forma ordenada
 */
void afegirDiaOrdenat(tSetmana *setmana, tDia dia);

/* Acció que copia el contingut d'un tDia origen (src)
 * a un tDia destí (dst)
 */
void copiarDia(tDia *dst, tDia src);

/* Acció que mostra per pantalla el contingut
 * de la taula tSetmana
 */
void mostrar(tSetmana setmana);

/* Programa principal */
int main(int argc, char **argv) {
    /* Definim les variables */
    tSetmana setmana, setmanaOrdenada;
    tDia dl, dm, dc, dj, dv, ds, dg;

    /* Inicialitzem les variables de tipus tDia */
    strcpy(dl.nom, "dilluns");
    dl.id = 1;
    strcpy(dm.nom, "dimarts");
    dm.id = 2;
    strcpy(dc.nom, "dimecres");
    dc.id = 3;
    strcpy(dj.nom, "dijous");
    dj.id = 4;
    strcpy(dv.nom, "divendres");
    dv.id = 5;
    strcpy(ds.nom, "dissabte");
    ds.id = 6;
    strcpy(dg.nom, "diumenge");
    dg.id = 7;
```

```
    /* Inicialitzem les variables de tipus
     * tSetmana
     */
    inicialitzar(&setmana);
    inicialitzar(&setmanaOrdenada);

    /* Afegim sense ordre les variables
     * tDia dins de les taules setmana
     * i setmanaOrdenada. La diferència
     * entre elles serà el mètode utilitzat
     * per inserir els tDia: mentre que
     * la inserció per setmana serà
     * l'habitual d'afegir com a darrer
     * element de la taula, per la
     * setmanaOrdenada utilitzarem l'acció
     * afegirDiaOrdenat()
     */
    afegirDia(&setmana, dj);
    afegirDia(&setmana, dl);
    afegirDia(&setmana, dc);
    afegirDia(&setmana, dg);
    afegirDia(&setmana, ds);
    afegirDia(&setmana, dm);
    afegirDia(&setmana, dv);
    afegirDiaOrdenat(&setmanaOrdenada, dj);
    afegirDiaOrdenat(&setmanaOrdenada, dl);
    afegirDiaOrdenat(&setmanaOrdenada, dc);
    afegirDiaOrdenat(&setmanaOrdenada, dg);
    afegirDiaOrdenat(&setmanaOrdenada, ds);
    afegirDiaOrdenat(&setmanaOrdenada, dm);
    afegirDiaOrdenat(&setmanaOrdenada, dv);

    printf("\nSetmana sense ordenar : \n");
    mostrar(setmana);
    printf("\nSetmana ordenada : \n");
    mostrar(setmanaOrdenada);
    return 0;
}

/* Implementació de les accions */

void inicialitzar(tSetmana *setmana) {
    setmana->numDies = 0;
}
```

```
void afegirDia(tSetmana *setmana, tDia dia) {
    /* numDies conté el número d'elements
     * de tipus tDia que conté la taula
     * tSetmana en cada moment
     */
    copiarDia(&setmana->dies[setmana->numDies], dia);

    /* Una vegada assignat un element nou
     * tDia a la taula, incrementem el valor
     * de numDies
     */
    setmana->numDies = setmana->numDies + 1;
}

/* Acció que afegeix un element de tipus tDia
 * a la taula tSetmana de forma ordenada
 */
void afegirDiaOrdenat(tSetmana *setmana, tDia dia) {

    int i, j;
    bool isAfegit;

    isAfegit = false;
    i = 0;

    /* Per fer l'ordenació s'utilitzarà el camp
     * id de tDia
     */
    for (i = 0; i < setmana->numDies && !isAfegit; i++) {

        /* Si l'id del tDia de la tSetmana > id del
         * paràmetre dia, a partir d'aquest punt
         * caldrà desplaçar les tuples tDia cap
         * a la dreta. L'objectiu és aconseguir un
         * lloc lliure, on s'afegirà el tDia dia
         * passat com a paràmetre a l'acció
         */
        if (setmana->dies[i].id > dia.id) {

            /* Desplaçament dels tDia cap a la dreta, a
             * partir de la posició on caldrà afegir
             * el paràmetre dia passat per valor
             */
            for (j = setmana->numDies; j > i; j--) {
                copiarDia(&setmana->dies[j], setmana->dies[j-1]);
```

```
            }

            /* S'afegeix el paràmetre dia a la
             * posició que li correspon, una vegada
             * desplaçats la resta de tuples tDia
             * cap a dreta
             */
            copiarDia(&setmana->dies[j], dia);
            setmana->numDies = setmana->numDies + 1;
            isAfegit = true;
        }
    }

    /* Si en aquest punt encara no s'ha
     * afegit el tDia, significa que ha d'anar
     * al final de tot de la taula tSetmana
     */
    if (!isAfegit) {
        copiarDia(&setmana->dies[i], dia);
        setmana->numDies = setmana->numDies + 1;
    }
}

void copiarDia(tDia *dst, tDia src) {
    dst->id = src.id;
    strcpy(dst->nom, src.nom);
}

void mostrar(tSetmana setmana) {
    int i;
    for (i = 0; i < setmana.numDies; i++) {
        printf("%s\n", setmana.dies[i].nom);
    }
}
```

L'execució del programa genera la següent sortida:

```
Setmana sense ordenar :
dijous
dilluns
dimecres
diumenge
```

```
dissabte
dimarts
divendres

Setmana ordenada :
dilluns
dimarts
dimecres
dijous
divendres
dissabte
diumenge
```

Chapter 12

PR2

12.1 Exemple: pilaCartes

A continuació s'exposa un exemple 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
```

S'han afegit comentaris dins el codi per facilitar la seva comprensió:

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

```c
/* 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

/* 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);
bool emptyStack(tStack s);
bool 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: s'ha 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;
}

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

bool 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 13

VMWare i CodeLite

13.1 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.

13.2 VirtualBox i requeriments de virtualització

Per poder utilitzar VirtualBox és necessari que disposi d'accés a l'opció de virtualització. Si no és així, es poden obtenir errors del tipus: **VT-X está deshabilitado en el BIOS para todos los CPUs**.

L'error es pot produir per vàries raons:

- La BIOS del PC té deshabilitada l'opció. Per solucionar-ho, cal entrar dins de la BIOS del PC i trobar l'opció d'activació de la virtualització (varia segons el fabricant): pot ser **VT-x**, **Virtualization Technology**,

VT-x/AMD-V, **Intel Virtual Technology**, **Tecnología de virtualización (VTx/VTd)**, etc.
- Tenim actiu un antivirus el qual té activada una virtualització pròpia que entra en conflicte amb la requerida per VirtualBox. En aquest cas, únicament cal desactivar l'opció de virtualització de l'antivirus.

13.3 Com instal · lar les Guest Additions

Les **Guest Additions** són una sèrie de drivers que milloren la interacció entre host i màquina virtual. Per instal · lar-les únicament cal fer:

1. Engegar la màquina virtual **FP20181**
2. Anar a la barra de menú superior -> **VirtualBox** -> **Devices** -> **Insert Guest Additions CD Image...** -> instal · larà una unitat de CD amb les **Guest Additions**.
3. Obrir un terminal des de **Lubuntu** -> **System Tools** -> **LXTerminal**
4. Dins del terminal **LXTerminal** executar el següent (la versió que s'estigui instal · lant pot ser més nova que la 6.0.12):

```
cd /media/uoc/VBox_GAs_6.0.12/
sudo sh ./VBoxLinuxAdditions.run
```

1. Reiniciar la **FP20181** per activar les **Guest Additions**.

13.4 Primers passos amb CodeLite

A les xWiki trobareu el mòdul **Introducció a l'entorn de programació CodeLite**, en el qual es detallen els passos a realitzar per preparar l'entorn.

A continuació es resumeixen una sèrie d'aspectes que es comenten a la xWiki i que són importants recordar:

- Un **workspace** de CodeLite és una agrupació de projectes.
- Únicament es pot tenir un **workspace** obert dins de CodeLite.
- Per crear un **workspace**: **CodeLite** -> **Workspace** -> **New Workspace...** -> **Workspace type:** `C++` -> **Workspace name**: el que correspongui; **Workspace Path**: `/home/uoc/Documents/codelite/workspaces/` (o qualsevol altre) -> fi
- Com que no es pot tenir més d'un **workspace** obert, no en podreu crear cap de nou si prèviament no tanqueu el workspace actiu. Si dins del menú de CodeLite veieu la opció **New Workspace** en gris i que no es pot seleccionar, significa que ja teniu un workspace obert. Per tancar-lo: **CodeLite** -> **Workspace** -> **Close Workspace**.
- Per afegir un projecte a un workspace: **CodeLite** -> **File** -> **New** -> **New Project** -> de tipus **Console**: `Simple executable (gcc)` ->

Project name: el que correspongui -> **Compiler**: `GCC`; **Debugger**: `GNU gdb debugger` -> fi

- El projecte que acabem de crear conté un programa `hello world` 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 els nostres programes en C (**no** els algorismes!).
- Si tenim més d'un projecte dins d'un workspace, la forma que tenim per indicar quin d'ells és el que està actiu és fent **doble clic** sobre el nom del projecte. Veureu que el nom queda remarcat en ***negreta i cursiva***: a partir d'aquest moment, aquest serà el projecte que compilarem i executarem des de les opcions del menú de CodeLite. Tot i que estiguem visualitzant per pantalla el codi d'un altre projecte, la compilació i execució sempre es farà del projecte actiu.
- Per mostrar la barra d'eines (icones) a la part superior: **CodeLite** -> **View** -> **Show toolbar**.
- La icona del **play** de color verd cap a la dreta de la toolbar és per **debuggar**, no per compilar.
- Per compilar i executar podem fer: **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.
- 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.

13.5 Com activar un projecte

Dins de CodeLite, a la part esquerra es mostren tots els projects que s'han creat al workspace. Si ens fixem en el nom de tots ells, veurem que un està remarcat en negreta; per exemple podem tenir:

- PAC01
- **PAC02**
- PAC03
- PAC04

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

Si es fa **doble clic** amb el ratolí sobre el nom del project **PAC04**, ara veurem:

- PAC01
- PAC02
- PAC03

- **PAC04**

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

13.6 Canviar idioma del teclat

Per canviar l'idioma del teclat a LLubuntu, cal clicar amb el botó dret del ratolí sobre la barra grisa superior -> **Add/Remove Panel Items** -> pestanya **Panel Applets** -> botó **Add** -> seleccionas el plugin **Keyboard Layout Handler** -> **Add** -> **Close**.

En aquests moments a la part superior dreta es mostrarà l'idioma definit per defecte pel teclat -> marcar sobre la bandera amb el botó dret -> **"Keyboard Layout Handler" settings** -> desmarcar l'opció **Keep system layout** -> es pot afegir l'idioma que es vulgui des del botó **Add**; es pot prioritzar un idioma o l'altre posant-lo en primera posició a la llista. Una vegada s'hagi guardat la configuració desitjada, ja haurà canviat la disposició del teclat al nou idioma.

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

13.7 Programa per defecte al crear un projecte

Cada vegada que es crea un nou projecte a CodeLite, per defecte sempre conté el codi del programa `hello world`:

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("hello world\n");
    return 0;
}
```

Per tant, quan volem crear el nostre propi programa, només cal esborrar el programa que crea per defecte i començar a codificar el teu programa nou.

Si malgrat tot no es vol que es mostri aquest codi cada vegada, podem modificar el codi per defecte associat als nous projectes:

- Editar l'arxiu `/usr/share/codelite/templates/projects/executable/main.cpp`, que és el que es carrega per defecte en crear cada nou projecte.
- Crear un nou template de CodeLite i utilitzar-lo en el moment de crear un nou projecte. La creació d'un nou templates es fa a partir d'un projecte que tinguem -> botó dret sobre el nom del projecte -> `Save as template...`.

13.8 Com fixar el kernel d'inici amb Lubuntu

Tot i ser una situació gens habitual, es pot donar el cas que Lubuntu només ens arranqui en un determinat kernel per problemes amb determinades targetes gràfiques.

Tot i que el kernel sempre es pot seleccionar en el moment d'inici de Lubuntu, podem fixar quin és el que volem utilitzar malgrat que aquest no sigui el més nou de tots. Una possible opció per fer-ho amb l'aplicació **Grub Customizer**; per instal · lar-la fem:

```
sudo add-apt-repository ppa:danielrichter2007/grub-customizer
```

Ens identifiquem amb la password de l'usuari **uoc**.

A continuació, actualitzem la llista de packages de tots els repositoris:

```
sudo apt-get update
sudo apt-get install grub-customizer
```

En aquest punt haurem instal · lat **Grub Customizer** dins de la màquina virtual **FP20181**: aquest programa ens permetrà confirmar exactament què hi ha definit a l'arranc grub de la màquina virtual. Per executar el programa: icona de **Lubuntu** —> **System Tools** —> **Grub Customizer** —> posar contrasenya **uoc** —> a la pestanya inicial **List configuration** es mostren tots els kernels de què disposa l'entorn en aquests moments.

Es recomanable fer una còpia prèvia de les PAC/PR que puguem tenir dins de la **FP20181** abans de canviar el kernel d'inici. Per canviar el kernel d'arranc per defecte: **Grub Customizer** -> pestanya **General Settings** -> com a **default entry**, seleccionar el kernel que funciona correctament d'entre tots els disponibles. Una vegada escollit, es prem el botó **Save** de la part superior esquerra.

Per aplicar el canvi que s'acaba de desar a l'arranc de Lubuntu cal obrir un terminal de Lubuntu des de **System Tools** -> **LXTerminal**, i des del terminal teclejar:

```
sudo update-grub
```

Una vegada hagi finalitzat el procés, cal realitzar un reboot de la **FP20181** per tal que agafi ara per defecte el kernel escollit.

Chapter 14

Altres

14.1 Bibliografia

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