Programació de Baix Nivell

—

Enginyeria de Sistemes TIC

Bloc Llenguatge C

Sebastià Vila-Marta

17 de febrer de 2026

Índex

Part I.

Material flipped classroom

1. Introducció

Aquest és el «manual» pel primer bloc de l'assignatura «Programació a BaixNivell». Aquest bloc gira al voltant de l'aprenentatge del llenguatge de programació C, però durant aquest viatge presenta i reforça molts altres coneixements de l'àmbit de la programació.

Aquest bloc està organitzat en sessions que tenen la mateixa estructura:

1. L'estudiant, individualment, treballa de forma autònoma un capítol (o part d'un capítol/s) del llibre seminal de `C` *The C programming language*, [6].

2. A la següent classe de teoria, el professor presenta alguns elements rellevants més o menys relacionats amb l'estudi que s'ha fet i es dedica un temps important per que els estudiants presentin dubtes i comentaris referents al tema estudiat. L'objectiu és treballar col·lectivament allò més punxegut del tema treballat individualment.

3. En la següent classe de problemes s'espera que l'estudiant, individualment, resolgui els problemes corresponents a la sessió, amb el suport del professorat de problemes/laboratori. Els problemes no resolts a classe és convenient resoldre'ls durant el temps d'estudi.

   La classe de problemes requerirà portar el vostre computador personal a classe convenientment configurat amb GNU/Linux i a punt per treballar.

Aquest document recull el material necessari per treballar aquestes sessions. Per cada sessió hi trobareu:

1. Els capítols del llibre que cal estudiar.

2. Unes preguntes motivadores per ajudar a treballar el tema d'estudi.

3. Una llista de problemes per a resoldre relacionats amb la sessió.

1.1. Recomanacions per a l'estudi

De cara a l'activitat d'estudi que implica cada sessió, us recomanem que considereu les següents recomanacions:

1. Llegiu sobre paper.

2. Llegiu conscientment, mirant d'entendre el que s'hi diu i prenent nota d'allò que no enteneu: conceptes, paraules concretes, etc.

3. Mireu de relacionar el que llegiu amb el que sabeu (de la vostra experiència amb Python, per exemple).

4. Rellegiu aquells passatges que trobeu més densos.

5. Mireu de llegir en un ambient sense distraccions que permeti concentrar-vos.

6. Feu servir les preguntes motivadores com element de reforç. Estan pensades per que, mirant d'entendre-les, forcin a repassar, repensar i relacionar amb altres coneixements allò que esteu estudiant.

1.2. Recomanacions per als problemes

Cada sessió conté també una llista de problemes de dificultat creixent que van entrenant en l'ús del llenguatge `C`. La idea és anar-los resolent, implementant i provant un darrera l'altre fins on sigui possible. Us recomanem que en el vostre temps d'estudi els acabeu de resoldre tots. L'objectiu final és anar aconseguint agilitat amb la sintaxi i les eines de treball relacionades amb el llenguatge `C`.

1.2.1. Material necessari

- El vostre computador personal funcional amb sistema operatiu GNU/Linux, preferentment sobre el hardware real (no virtualitzat).

- Els paquets `emacs`, `gcc`, `libc6-dev` instal·lats.

1.2.2. Control de versions

Aquesta pràctica i totes les que segueixen cal desenvolupar-les amb el suport —obligatori— d'un sistema de control de versions. El sistema a emprar és `subversion`, [3]. Necessàriament cal hostatjar el dipòsit de versions a `https://escriny.epsem.upc.edu`.

La manera d'organitzar el dipòsit de versions és crear un sol projecte i tenir un directori específic per cada pràctica. L'estructura de directoris que n'ha de resultar ha de ser similar a la que s'observa a la figura 1.1. Recordeu que només se sotmeten a la disciplina del control de versions els fitxers font del projecte i altres fitxers que defineixen el projecte com ara els `Makefile`.

Doneu d'alta un projecte i el corresponent dipòsit de versions a `escriny` per a poder gestionar les pràctiques.

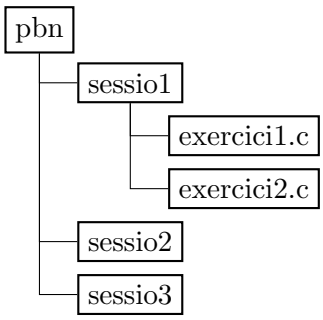

Figura 1.1.: Estructura del dipòsit de versions pels problemes.

2. Sessió 1

2.1. Text d'estudi

Estudieu les següents parts del llibre:

- Chapter 1: A Tutorial Introduction (llevat de l'apartat 1.10)

2.2. Preguntes per pensar

1. Quin paper juga el ;? És un separador de sentència o un final de sentència?
2. Quan posem claus {} i per quina raó les posem? Quin sentit gramatical tenen?
3. Quina diferència hi ha entre aquestes dues constants: `'a'` i `"a"`?
4. Quina diferència hi ha entre aquestes dues constants. `3` i `3.0`?
5. A la pàgina 21 hi ha una regla sintàctica per a la sentència condicional. D'acord amb aquesta regla, els parèntesis que hi ha darrera l'**if**, són obligats? formen part de l'expressió?
6. Què és una expressió? i una sentència?
7. Què és una funció? Com la definiries?
8. Quines diferències hi ha entre una taula (array en anglès) i una llista?
9. Per què la funció `getchar()` retorna un **int**?
10. Què és un argument (o actual parameter) i què és un paràmetre (o formal parameter).
11. Com funciona el mecanisme de pas de paràmetres per valor?
12. Què és un prototip? per què deuen existir?
13. Què n'opines de l'estil de programació d'aquest capítol?

2.3. Exercicis pràctics

Exercici 2.1 Dissenyeu i implementeu un programa en `C99` que escriu pel canal de sortida la frase `"Mort! qui t'ha mort?"`.

Exercici 2.2 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada un enter n seguit d'un caràcter `c` i escriu pel canal de sortida el caràcter repetit n vegades.

Exercici 2.3 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada una frase acabada en el caràcter punt i escriu pel canal de sortida quantes vegades apareix la lletra `'a'`.

Exercici 2.4 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada un byte en format hexadecimal i n'escriu les representacions en base 2, 8 i 10 pel canal de sortida. El programa cal que es digui `converteix`.

L'exercici és més interessant si en feu dues versions:

1. Una en que el càlcul de la nova representació l'implementeu «a pèl» en el mateix programa.
2. Una altra en que aprofiteu tant com sigui possible les conversions de format que ofereix la funció `printf()` de la llibreria de `C`.

Exercici 2.5 Dissenyeu un programa que llegeix del canal d'entrada un real δ i escriu ben formatada una taula de conversió de graus Cèlsius a graus Fahrengheit entre 0 °C i 100 °C.

Exercici 2.6 Dissenyeu i implementeu un programa que llegeix un text acabat en `EOF` i l'escriu comprimint els espais. És a dir, quan troba més d'un espai seguit els substitueix per un sol espai.

Exercici 2.7 Dissenyeu i implementeu un programa que llegeix un text acabat en `EOF` i escriu un histograma de les longituds de les línies. És a dir, per cada línia llegida escriu una línia amb la longitud de la línia llegida seguida de tants asteriscs com caràcters tenia la línia.

Per exemple, si es llegeix la línia

```
In Springfield, they're eating the cats!
```

hauria d'escriure

```
40  |****************************************
```

Exercici 2.8 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada els coeficients d'una equació de segon grau i escriu pel canal de sortida la seva solució.

Exercici 2.9 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada un preu en euros i escriu pel canal de sortida el desglòs mínim en moneda que correspon al preu.

3. Sessio 2

3.1. Text d'estudi

Estudieu les següents parts del llibre:

- Chapter 2: Types, Operators and Expressions

3.2. Guia de lectura

1. Què és un tipus de dades? I què és un valor d'un tipus? I la representació d'un valor? Tots els valors d'un tipus tenen la mateixa representació? Hi ha una sola representació possible d'un valor d'un tipus?

2. Tipus i representació. Quina relació tenen? Són rellevants per l'etapa de disseny d'un programa? Tenen relació amb el concepte d'objecte (en el sentit de l'orientació a objectes)?

3. Constants vs variables amb qualificador **const**. Quines similituds i diferències tenen?

4. Què significa que quelcom sigui «implementation-defined»? Podem basar un programa en quelcom que és «implementation-defined»? Per què?

5. Què és la precedència —també dita prioritat— i l'associativitat d'un operador? Per què existeixen?

6. Què és un operador unari?

7. Per què l'operador de postincrement es diu que actua via efecte lateral?

8. Per què l'operador >> sobre tipus integrals amb signe generalment comporta «extensió de signe» (*arithmetic shift*)?

9. Per què l'assignació és un operador a C en comptes de ser una sentència com habitualment?

10. Quina diferència hi ha entre les operacions `&` i `&&` ?

11. Quina diferència hi ha entre un operador i una operació?

12. Què és un *cast*?

13. Què és un operador relacional? I un operador lògic?

14. Quina és el sentit de les operacions *bitwise*?

3.3. Exercicis pràctics

EXERCICI 3.1 Afegiu opcions al programa resultant de l'exercici 2.4 de manera que `converteix -b` converteixi només a binari, `converteix -o` a octal i `converteix -d` a decimal.

Aquest exercici requereix investigar com es traslladen a un programa `C` els paràmetres amb que es crida des de la shell de Unix.

EXERCICI 3.2 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada una paraula de 2 B en hexadecimal i escriu per la sortida la mateixa paraula després d'haver forçat sengles zeros els bits de més i menys pes. En tot moment la paraula caldrà emmagatzemar-la com a tal i no com una cadena de caràcters o com una taula.

Per exemple, si l'entrada és `0aaa` la sortida hauria de ser `8aab`.

EXERCICI 3.3 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada una paraula de 2 B en hexadecimal i escriu per la sortida el resultat d'extreure el byte que va dels bits 4 al 11 (el bit de menys pes és el bit 0). En tot moment la paraula i el byte caldrà emmagatzemar-la com a tal i no com una cadena de caràcters o com una taula.

EXERCICI 3.4 Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada una seqüència binària codificada en hexadecimal i acabada en un byte 0 i escriu pel canal de sortida el nombre de bits amb valor 1 d'aquesta seqüència.

EXERCICI 3.5 Considereu que un senyal analògic entre $-1{,}0$ V i $1{,}0$ V el mostregeu a $100{,}0$ Hz i codifiqueu digitalment el resultat sobre 1 B en complement a 2. Una cadena de 100 B, doncs, representa 1 s d'aquest senyal. Assumiu que treballem sempre amb segments d'1 s que els representem com a cadenes hexadecimals.

Dissenyeu i implementeu un programa en `C99` que llegeix pel canal d'entrada un segment de senyal i escriu pel canal de sortida un altre senyal de la mateixa natura que l'anterior amb un pols d'amplitud 1 V i 20 ms de durada quan el primer senyal fa un pas per zero.

EXERCICI 3.6 Dissenyeu un programa que llegeix un enter k d'un byte de longitud, seguidament en calcula $-k$ assumint que es representa en complement a 2 i sense fer servir operacions aritmètiques. Finalment escriu el resultat.

L'objectiu és manipular a nivell de bit la representació en complement a 2 d'un enter.

EXERCICI 3.7 Dissenyeu un programa que determini els valors mínim i màxim del domini dels tipus següents: **short**, **unsigned short**, **int** i **long**. Feu-ho de dues maneres:

1. Fent servir `<limits.h>`.
2. Calculant-los mitjançant operacions bit a bit (desplaçaments i complements).

EXERCICI 3.8 Escriviu una funció amb prototip **int** es_potencia2(**unsigned** x) que retorni 1 si x és una potència de 2 i 0 en cas contrari. Feu servir només operadors bit a bit i comparacions. Proveu la funció.

Part II.

Notes sobre C

4. Tutorial C99

El llenguatge de programació C té un paper fonamental en aquesta assignatura. Hi ha molta i molt bona literatura sobre el llenguatge C, especialment en la seva variant C89.

Com a bibliografia recomanem [6], que hauria de ser un dels llibres de capçalera. En segona instància també els llibres lliures [1, 2] i els apunts [7] poden ser d'utilitat.

A mode de xuletari, el tríptic [9], és d'allò més pràctic.

Finalment, molts llibres sobre la disciplina dediquen capítols a introduït el llenguatge C. Aquest és el cas de [8]

En aquest apèndix només hi trobareu un resum molt breu i sintètic dels aspectes més interessants en l'àmbit d'aquest document que incorpora l'estàndard C99. Les raons i la lògica dels canvis que incorpora C99 respecte C89 estan documentades a [5].

4.1. Primer exemple

Un programa codificat en C és una col·lecció de funcions una de les quals té un nom privilegiat, main(), i actua com a programa principal. La figura 4.1 és un programa en C que conté una funció i un programa principal.

Si aquest programa el contingués el fitxer senzill.c, el compilaríem i muntaríem amb la comanda:

```
$ gcc -std=c99 -o senzill senzill.c
```

i tot seguit provaríem d'executar-lo fent:

```
$ ./senzill
```

L'executable cal prefixar-lo per nom del directori atès que les shells de UNIX només cerquen executables en els directoris que indica la variable d'entorn PATH. Habitualment el directori de treball no forma part d'aquesta col·lecció de directoris. Per tant cal explicitar on és el fitxer executable. En aquest cas usem un camí relatiu.

4.2. Variables i tipus escalars

4.2.1. Variables i tipus

Les declaracions de variables tenen aquest aspecte:

```
int x, y;
```

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

int dobla(int x) {
  int r;

  r = x * 2;
  return r;
}

int main() {
  int v, n;

  n = scanf("%d", &v);
  assert(n == 1);
  printf("%d\n", dobla(v));

  return 0;
}
```

Programa 4.1: Programa senzill en C99

i poden ocórrer en qualsevol lloc del codi[1]. Com es veu, primer s'escriu el tipus de dades i a continuació una llista d'identificadors de variable.

Els tipus de dades més corrents són els següents:

Tipus	Significat	Exemple
bool	Un booleà. Cal incloure prèviament el header stdbool.h.	`#include <stdbool.h>` `bool b; b = false;`
char	Un caràcter (no una cadena!)	**char** c; c = 'a';
short	Un enter "curt". Habitualment 16 bit.	**short** x; x = 67;
int	Un enter habitualment de 32 bit	**int** i; i = −12;
long	Un enter llarg. Habitualment de 64 bit	**long** l; l = 3L;
float	Un real de precisió simple	**float** x; x = 23.45; x = 2.0e−13;

Les mides dels tipus depenen de la plataforma i el compilador. Per tant no són portables. El header limits.h defineix una sèrie de constants que permeten conèixer els límits de cada tipus de dades.

La taula 4.1 mostra la classificació habitual dels tipus escalars de C99.

Els tipus disposen d'operació de conversió de tipus explícita (*cast*):

[1] És una característica apareguda a C99

Classificació				Tipus
Integrals	Booleà			bool
	Caràcter			**char**
	Enter	Ordinari	Amb signe	**short**
				int
				long
			Sense signe	**unsigned short**
				unsigned int
				unsigned long
		Mida fixa	Amb signe	int8_t
				int16_t
				int32_t
			Sense signe	uint8_t
				uint16_t
				uint32_t
Reals	Flotant			**float**
				double

Taula 4.1.: Classificació dels tipus escalars de C

```
int i;
short s;

i = 78;
s = (short)i;
```

De la mateixa manera, existeixen un conjunt de regles de conversió que s'apliquen de forma automàtica (*coercion*)[2]. En general un tipus promociona automàticament a un tipus de grandària superior. Cal, però, tenir cura quan es barregen tipus amb i sense signe. És una font important de problemes i cal anar amb peus de plom.

Tots els tipus enters disposen de la versió sense signe. El tipus sense signe associat a un tipus amb signe es construeix prefixant amb **unsigned**. També es poden definir constants sense signe usant el sufix `U` a tal efecte. Per exemple:

```
unsigned int i;

i = 34U;
```

Sorprenentment, el tipus **char** es considera amb signe i, per tant, existeix el tipus **unsigned char**.

També és possible usar constants enteres escrites en base 2[3], 8 i 16:

```
unsigned int x;
```

[2]Noteu la diferència de significat entre *cast* i *coercion*.
[3]A partir de `C99`.

```
x = 0b010; // binari
x = 0347; // octal
x = 0x3af; // hexadecimal
```

Per evitar la manca de portabilitat dels tipus de dades pel que fa a la seva longitud, C99 defineix un conjunt de tipus amb la mida ben definida. Per usar-los és imprescindible incloure el header stdint.h. Els tipus en qüestió són uint8_t, uint16_t, uint32_t i els seus corresponents amb signe int8_t, int16_t i int32_t, que com el seu nom indica corresponen a mides de 8 bit, 16 bit i 32 bit respectivament. Cal usar aquests tipus quan ens cal representar quelcom que ha de tenir una longitud fixada, per exemple un port o un registre.

La funció predefinida **sizeof** permet consultar en temps d'execució la mida de qualsevol tipus. El resultat es dona usant com a unitat la mida d'un **char**. Així, per exemple, pot succeir que **sizeof**(**int**) sigui 4, indicant que un **int** ocupa 4 **char**'s. La mida en bits d'un **char** es pot consultar emprant la constant de limits.h anomenada CHAR_BIT. Molt sovint CHAR_BIT==8.

4.2.2. Operacions

A C l'assignació es considera una operació que, per efecte lateral, modifica el contingut d'una variable. Així, per exemple, x=3 és una operació que modifica el valor de la variable x i, a la vegada, val 3. D'aquesta forma, es pot escriure y=(x=3) o, traient els parèntesis y=x=3, que té l'efecte d'assignar el valor 3 tant a x com a y.

Les operacions sobre el tipus bool són les habituals:

Operació	Significat
`||`	Disjunció booleana (OR).
`&&`	Conjunció booleana (AND).
`!`	Negació booleana (NOT).
`==`	Igualtat.
`!=`	Diferència.

Sobre els tipus integrals (enters), les operacions es classifiquen en:

- Operacions aritmètiques:

Operació	Significat
`+`	Suma.
`*`	Producte.
`-`	Resta o canvi de signe.
`/`	Divisió entera quan ambdós operands són enters.
`%`	Mòdul.

- Operacions booleanes:

Operació	Significat
==	Igualtat.
!=	Diferència.
>, >=, <, <=	Comparació

- Operacions bit a bit:

Operació	Significat
>>	Shift dreta. Amb extensió de signe si el tipus és signed.
<<	Shift esquerra. Amb extensió de signe si el tipus és signed.
~	Complement.
&	AND bit a bit.
|	OR bit a bit.
^	OR exclusiva bit a bit.

Els enters i booleans disposen a més d'operacions de modificació, que combinen l'assignació i una operació específica. Per exemple, les dues operacions següents són equivalents:

```
x *= 2;
x = x * 2;
```

De manera similar, es disposa d'operacions d'auto increment/decrement. Així, i++ significa: "avalua i i, posteriorment, incrementa el seu valor". Això es coneix com post-increment. De manera simètrica, el pre-increment ++i significa "incrementa i i avalua-la". D'acord amb això el següent fragment de programa és tal que en acabar a==128, ra==128, b==128 i rb==127.

```
int a, b, ra, rb;

a = b = 127;
ra = ++a;
rb = b++;
```

També disposem d'operadors de pre i post-decrement amb la sintaxi esperable: j−− i −−j.

En el cas dels reals en coma flotant, les operacions aritmètiques bàsiques són:

Operació	Significat
+	Suma.
*	Producte.
-	Resta o canvi de signe.
/	Divisió real quan un dels operands és real.

A banda de les operacions elementals, si s'inclou el header math.h es pot accedir a una col·lecció d'operacions i funcions més àmplia. Aquesta col·lecció inclou funcions com:

Operació	Significat
`sin`, `cos`, `tan`	Funcions trigonomètriques.
`sqrt`	Arrel quadrada.
`exp`	Exponenciació.
`log`	Logaritme natural.

En qualsevol referència com ara la Viquipèdia i, naturalment l'estàndard [4], trobareu la documentació completa.

4.2.3. Algunes transgressions

De fet a `C` és corrent, però delicat, tractar indiscriminadament qualsevol tipus enter com si es tractés d'un booleà i confondre deliberadament caràcters i enters. Les conversions entre tipus faciliten aquesta pràctica.

Pel que fa als booleans, qualsevol tipus integral pot ser considerat un booleà sota la premissa de que el valor 0 cal entendre'l com a false i la resta de valors com a true. Així, el següent exemple és plenament vàlid:

```
int i;
bool b;

b = false; i = 33;
if (b || i)
  printf("Aja!");
```

Seguint amb les transgressions, també es comú entendre els **char** com a enters «petits». En el següent exemple, el programador tracta la variable c sense donar-li en cap moment el significat de caràcter:

```
char c;
int i;

c = 34;
i += c
```

En casos com aquest és més interessant usar una variable de tipus int8_t en comptes de **char**.

4.2.4. Prioritat i associativitat dels operadors

Atesa la quantitat d'operadors diferents emprats a `C`, la interpretació de les expressions pot ser difícil i cal tenir a mà la taula de prioritats i associativitats. La taula 4.2 mostra aquesta informació.

Prioritat	Operador	Associativitat
1	++, −− (post) [], () ., −>	LR
2	++, −− (pre) +, −, *, & (unaris) (cast), ~, !	RL
3	*, /, % (multiplicatives) +, − (additives)	LR
4	>>, <<	
5	>, >=, <, <=	
6	==, !=	
7	&	
8	^	
9	|	
10	&&	
11	||	
12	?: (expr. cond.)	RL
13	=, +=, *=, etc.	
14	, (coma)	LR

Taula 4.2.: Taula de prioritats i associativitat dels operadors C.

4.3. Implementació de mòduls

4.3.1. Esquema bàsic

Aquesta secció descriu la forma d'implementar el concepte de mòdul usat en el disseny d'una aplicació a l'entorn de `C`. Tot i que hi ha diversos esquemes possibles de treball, aquí se'n presenta només un d'específic que és prou general i ortogonal.

La idea és aprofitar la possibilitat de fer compilació separada per implementar mòduls seguint un esquema d'organització ben establert. Entenem que un mòdul és un contenidor de funcions, constants i tipus de dades que poden ser usats per altres mòduls. Addicionalment un mòdul defineix un espai de noms privat que permet ocultar detalls d'implementació a la resta dels mòduls que composen un projecte. Un mòdul, doncs, té una part pública i una part privada.

L'organització que es proposa implementa un mòdul sobre dos fitxers, un header i un fitxer de codi c. El primer conté la part pública del mòdul mentre que els segon conté

la part privada. Vegem-ho en una sèrie d'exemples.

Suposem que el mòdul mod1, per exemple, és un mòdul funcional que implementa dues funcions públiques: inc() i dec(). La implementació d'aquestes funcions és privada. Aleshores la forma d'implementar el mòdul és escriure dos fitxers. El primer, un fitxer de headers l'anomenem mod1.h i té aquesta forma:

```
#ifndef MOD1_H
#define MOD1_H

void inc(int *const i);
void dec(int *const i);

#endif
```

L'estructura d'un header és sempre la mateixa. Primer hi ha un parèntesi format per les sentències de preprocessador **#ifndef** i **#endif** que impossibilita una doble inclusió d'aquest header en un fitxer. Tot i que pot semblar excessiu, és possible que es doni una doble inclusió atès que en un fitxer determinat es pot incloure indirectament el mateix header per diversos camins i ser difícil de detectar. Aquest parèntesi, que cal escriure sempre, evita aquest problema. Noteu que el símbol MOD1_H ha de ser particular de cada mòdul. Així, si el mòdul es digués az34, per exemple, usaríem el símbol AZ34_H.

A l'interior del parèntesi, en aquest cas, el mòdul conté les capçaleres de les funcions públiques. Noteu que les capçaleres són imprescindibles per a que el compilador pugui saber si les crides a aquestes funcions són correctes.

Pel que fa a la part privada del mòdul, la escriuríem en el fitxer mod1.c, que tindria aquest aspecte:

```
#include "mod1.h"

static int suma(int k, int v) {
  return k + v;
}

void inc(int *const i) {
  *i = suma(*i, 1);
}

void dec(int *const i) {
  *i = suma(*i, -1);
}
```

Noteu que, de forma una mica artificial, s'ha usat una funció privada del mòdul per implementar dec() i inc(). Aquesta funció, suma(), s'a declarat com **static** per fer-la privada i, naturalment, el seu prototip no apareix al fitxer mod1.h.

Si des de la implementació d'un segon mòdul, posem que és modu.c, es volen usar les funcions de mod1, cal simplement afegit l'include escaient i cridar-les on convingui:

```
#include <stdio.h>
#include "mod1.h"

int main() {
  int v = 3;

  inc(&v);
  printf("%d\n", v);

  return 0;
}
```

Per compilar aquest exemplet simplement caldria fer:

```
$ gcc -std=c99 -I. -c mod1.c
$ gcc -std=c99 -I. -c modu.c
$ gcc -std=c99 -o modu modu.o mod1.o
```

o bé, en una sola ordre fent:

```
$ gcc -std=c99 -I. -o modu modu.c mod1.c
```

4.3.2. Singleton

Un *singleton* en la terminologia d'orientació a objectes és un objecte del que només n'existeix una sola instància en una aplicació. Molt sovint s'usen els mòduls per a implementar aquest concepte. Imagineu que en una aplicació feta de diversos mòduls hi ha un únic comptador que s'incrementa de 5 en 5 i es decrementa de 3 en tres. Del comptador la única informació útil és saber si és positiu o negatiu. Aquest comptador es pot implementar com un mòdul que exporta les operacions indicades i que *encapsula* la representació del comptador de manera privada.

La implementació del header seria la següent si assumim que correspon al fitxer `compt.h`:

```
#ifndef COMPT_H
#define COMPT_H

#include <stdbool.h>

void inc(void);
void dec(void);
bool is_positive(void);

#endif
```

Noteu:

- Només conté els prototips de les funcions públiques però cap referència a la implementació del comptador.
- Inclou el header `stdbool` atès que s'usa el tipus bool.

La implementació, desada en el fitxer `compt.c` seria la següent:

```
#include "compt.h"

static int comptador = 0;

void inc(void) {
  comptador += 5;
}

void dec(void) {
  comptador -= 3;
}

bool is_positive(void) {
  return comptador >= 0;
}
```

Noteu l'ús de **static** per indicar que la variable comptador és privada i no pot ser usada des de cap altre mòdul.

Usant aquest mòdul assegureu que no pot haver-hi cap altra còpia del comptador en un programa.

4.3.3. Classes

A vegades els mòduls s'usen per implementar estructures similars en certa manera a una classe: un tipus de dades acompanyat d'un conjunt d'operacions. En aquests casos tant el tipus de dades com les seves operacions són públiques. Imaginem que volem implementar un tipus que permeti definir piles d'enters de mida no afitada. Aleshores el header del mòdul, que podríem anomenar `stack.h`, seria similar a:

```
#ifndef STACK_H
#define STACK_H

typedef void *stack;

stack push(stack p, int i);
stack pop(stack p);
int top(stack p);
stack empty(void);

#endif
```

i la seva implementació, sense tenir en compte possibles excepcions, correspondria a:

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

typedef struct sc {
  int v;
  struct sc *next;
} cell_t;

stack empty(void) {
  return NULL;
}

stack push(stack p, int i) {
  cell_t *c = malloc(sizeof(cell_t));
  c->v = i;
  c->next = p;
  return c;
}

stack pop(stack p) {
  cell_t *c = ((cell_t *)p) -> next;
  free(p);
  return c;
}

int top(stack p) {
  return ((cell_t *)p) -> v;
}
```

D'aquesta manera, qualsevol altre mòdul podria definir variables de tipus stack i operar amb elles fent:

```
#include "stack.h"

...
stack s = empty();
s = push(s,30);
s = push(s, 10);
printf("%d\n", top(s));
s = pop(s)
...
```

Bibliografia

[1] Mike Banahan, Declan Brady i Mark Doran. *The C Book*. Angl. 2a ed. Addison-Wesley, Pearson Education, 1991. URL: http://publications.gbdirect.co.uk/c_book (cons. 08-02-2017).

[2] Mark Burgess i Ron Hale-Evans. *The GNU C Programming Tutorial*. Angl. 4.1. 2002. 290 pàg. URL: http://www.crasseux.com/books/ctut.pdf (cons. 08-02-2017).

[3] Ben Collins-Sussman, Brian W. Fitzpatrick i C. Michael Pilato. *Version Control with Subversion*. Angl. Vers. 1.7. 2011. URL: http://svnbook.red-bean.com/en/1.7/svn-book.pdf (cons. 20-02-2019).

[4] Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 22, Working Group 14. *Draft of the C99 standard with corrigenda TC1, TC2, and TC3 included*. Angl. Draft standart ISO/IEC 9899:TC3. ISO/IEC, 2007. 519 pàg. URL: http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf (cons. 08-02-2017).

[5] Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 22, Working Group 14. *Rationale for International Standard – Programming Languages – C*. C99. Angl. Rationale. Vers. Revision 5.10. ISO/IEC, 2003. 224 pàg. URL: http://www.open-std.org/jtc1/sc22/wg14/www/C99RationaleV5.10.pdf (cons. 08-02-2018).

[6] Brian Kernighan i Denis Ritchie. *The C Programming Language*. Angl. 2a ed. Addison-Wesley, Pearson Education, 1988. 274 pàg. ISBN: 9780131103627.

[7] Nick Parlante. *Essential C*. Angl. 2003. 45 pàg. URL: http://cslibrary.stanford.edu/101/EssentialC.pdf (cons. 08-02-2017).

[8] David Russell. *Introduction to Embedded Systems. Using ANSI C and the Arduino Development Environment*. Angl. Synthesis Lectures on Digital Circuits and Systems 30. Morgan & Claypool Publishers, 2010. 255 pàg. ISBN: 9781608454983. DOI: 10.2200/S00291ED1V01Y201007DCS030.

[9] Joseph H. Silverman. *C Reference Card (ANSI)*. Angl. 1999. 2 pàg. URL: http://www.math.brown.edu/~jhs/ReferenceCards/CRefCard.v2.2.pdf (cons. 08-02-2017).