Programació Fonamental: Pràctiques de Laboratori

Lluís Solano Albajés
Núria Pla Garcia
Antoni Soto Riera
Sebastià Vila Marta

13 d'octubre de 2010

Índex

Copyright

Capítol 1

Introducció

Aquest llibre s'adreça a estudiants d'assignatures d'introducció a la programació dins d'estudis universitaris on la informàtica no és l'eix central. En particular, pretén cobrir les hores de docència dedicades a pràctiques de laboratori.

L'elaboració de material per a pràctiques de laboratori fa imprescindible prendre algunes decisions sobre el material i les eines disponibles en l'entorn del laboratori d'informàtica. Hem procurat prendre les decisions mínimes, de manera que resulti relativament simple adaptar aquest llibre a altres entorns. Hem suposat que el laboratori d'informàtica disposa d'una o més aules amb ordinadors connectats en xarxa. Suposem també que executen el sistema operatiu `UNIX` o `GNU/Linux`. Només es faran servir dues eines: un editor de textos i un compilador. L'editor emprat s'anomena `EMACS` i el compilador, `GCC`. Són programes produits per la *Free Software Foundation* dins del projecte *GNU*. Han estat seleccionats per tres raons: la seva molt alta qualitat, estan disponibles sota els sistemes operatius més famosos —`Unix`, `VMS`, `OS/2`, `WindowsNT` ...— i es distribueixen sota llicències de programari lliure. Finalment, el disseny dels algorismes es fa en una notació algorísmica que posteriorment es tradueix al llenguatge de programació `C`. Pretenem així evitar que la complexitat d'un llenguatge de programació distregui l'atenció de l'alumne durant la fase de disseny dels algorismes. S'ha triat el llenguatge de programació `C` per la seva àmplia difusió. L'apèndix D descriu la sintaxi de la notació algorísmica emprada i dóna les pautes que cal seguir per a la traducció d'un algorisme a `C`.

Cada un dels capítols correspon a una sessió de laboratori. Han estat calculats per a sessions de dues hores, tot i que és possible adaptar les pàctiques, combinant-ne més d'una o simplificant-les, a altres durades de les sessions sense massa esforç.

La major part de les pràctiques s'han de fer en tres etapes. Cal començar amb una lectura del capítol corresponent i fer émfasi en els apartats introductoris. En aquesta etapa també es podran resoldre els exercicis proposats que no requereixin l'ús de l'ordinador. La segona etapa és la sessió de laboratori i s'ha de dur a terme a l'aula informàtica, davant d'un ordinador, de manera que, simultàniament amb la lectura de l'apartat *"Guió de treball"*, pugui provar-se el que s'explica. Molt sovint es proposen exercicis complementaris que poden ser resolts en aquest moment. La darrera etapa consisteix a resoldre els exercicis proposats que requereixin dades obtingudes durant la sessió de laboratori.

Podem agrupar els capítols d'aquest llibre en tres blocs. El primer comprèn els capítols 2, 3, 4, 4.3.4, 6 i 7. En ells es tracten els conceptes més bàsics relacionats amb el sistema operatiu, l'editor i el compilador. Les pràctiques corresponents a aquests capítols s'han de fer en l'ordre en què es presenten en el llibre ja que cada un usa conceptes apareguts en els anteriors. El segon bloc comprèn els capítols 8 i 9, que presenten conceptes més avançats relacionats amb la compilació separada i l'ús de llibreries. Per tal de comprendre els conceptes exposats en aquests capítols és imprescindible haver fet les pràctiques del primer bloc. En el capítol de llibreries s'usa el concepte de compilació separada; per tant, les pràctiques corresponents s'han de dur a terme en l'ordre en què apareixen en aquest llibre. El tercer bloc, està format pels capítols 10, 11 i 12, que tracten sobre l'ús de la memòria dinàmica, la complexitat dels algorismes i els fitxers de

text, respectivament. El contingut dels capítols d'aquest bloc no té cap relació entre si i s'hi usen conceptes apareguts en el primer bloc. També és convenient haver fet les pràctiques del segon bloc, però no és imprescindible.

L'apèndix A conté els llistats de tots el programes d'aquest llibre. Als apèndix B i C trobareu uns quadres que, a mode de resum, exposen les principals comandes del sistema operatiu i de l'editor emprades. A l'apèndix D es mostra la sintaxi de la notació algorísmica i les pautes de traducció a `C`.

Finalment, cal dir que també s'hi inclou un índex alfabètic, que serà d'inestimable ajuda quan aquest llibre s'utilitzi com a manual de referència. Amb l'índex, la localització en el text dels punts on es parla d'un cert tema és ràpida i còmoda.

Capítol 2

Introducció al computador

2.1 Material necessari

- Disquet de 3.5 polzades i alta densitat (1.44 Mb). Procureu que el disc sigui de bona qualitat. Heu de pensar el resultat de moltes hores de la vostra feina s'emmagatzemarà en aquest mitjà. Si la qualitat no és prou bona *podeu perdre tota la feina!*

2.2 Objectius

- Conèixer com és un laboratori de càlcul i com funciona.
- Saber com cal tractar un computador.
- Aprendre a establir una sessió de treball en una xarxa de computadors.
- Aprendre què és un sistema operatiu.
- Aprendre algunes comandes fonamentals d'un sistema operatiu.

2.3 Guió de treball

2.3.1 El computador

Les pràctiques d'informàtica es duen a terme en el laboratori d'informàtica. El laboratori d'informàtica disposa de diverses aules d'informàtica amb computadors i altres serveis. Les aules estan gestionades pels operadors. Un operador és una persona que vetlla pel funcionament correcte del servei d'informàtica. La seva responsabilitat inclou l'estat físic dels computadors (teclats en bon estat, màquines a punt, impressora en bon estat, etc.) i també altres qüestions com la gestió dels llistats. Així doncs, aquestes són les persones a qui heu de dirigir les vostres preguntes quan alguna cosa de la sala no funcioni correctament o bé tengueu dubtes de com utilitzar el material de la sala. La vostra responsabilitat inclou el tracte correcte del material i saber emprar adequadament les aplicacions informàtiques de què es disposa.

L'element fonamental de l'aula d'informàtica són els computadors. Cada lloc de treball està format pels elements següents (vegeu la figura 2.1):

- La unitat central. És la capsa que conté la majoria de components electrònics del computador. Aquesta capsa habitualment serveix de suport per a la pantalla i conté les unitats lectores de disc. No requereix una cura especial, llevat amb les connexions del darrere, que són molt sensibles a les estirades i moviments dels cables.

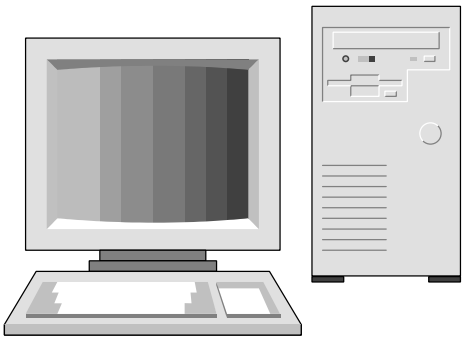

Figura 2.1: Un computador modern

- La pantalla. La pantalla és la part del computador que té aspecte de televisor. A través seu el computador comunica resultats. Essencialment la pantalla és un monitor de televisió. Per tant també té controls d'intensitat i contrast que cal regular fins a obtenir una visió adequada. Recordeu, però, que una intensitat exagerada provoca una degeneració molt ràpida de les pantalles i cal evitar-la.

- El teclat. Aquesta és la part del computador que recorda una màquina d'escriure. La seva funció és permetre el subministrament d'informació al computador. A més de les tecles que hi ha en una màquina d'escriure, disposa de tecles específiques que anirem comentant. És el dispositiu més delicat del computador. Vessar algun producte al damunt o deixar-hi caure alguna cosa com molles o terra pot fer malbé un teclat. Per tant, *cal abstenir-se de menjar, beure o fer qualsevol altra activitat no necessària davant d'un computador.* També cal anar amb compte a l'hora de picar una tecla. Les tecles estan construïdes per tal que el seu moviment sigui suau i precís. Si les maltractem polsant amb força exagerada o bé picant damunt el teclat, en reduïm dràsticament la vida útil.

Els computadors de la sala d'usuaris estan connectats a una xarxa. Una xarxa de computadors és un conjunt de computadors connectats entre si amb un mitjà físic que comparteixen recursos. Usualment aquests computadors comparteixen unitats de disc i impressores. D'aquesta manera s'evita haver de duplicar tot el conjunt de programes comuns i es pot treure un major rendiment de la impressora.

2.3.2 La sessió de treball

S'anomena sessió al conjunt d'activitats que es duen a terme en un computador durant l'interval de temps que va des que comencem a treballar fins que acabem. Alguns sistemes tenen control d'accés. Això vol dir que, per poder començar una sessió, cal aconseguir una autorització després d'identificar-se. Aquest és el cas de la majoria de sales d'usuaris. El procés d'identificació previ a la sessió s'anomena *login* (connexió). Quan la sessió ha acabat, cal indicar-ho explícitament. El procés mitjançant es fa s'anomena *logout* (desconnexió).

Per a dur a terme un *login* cal conèixer, en general, dues informacions: el nom d'usuari (*username*) i la paraula clau (*password*). El primer identifica qui vol començar una sessió i el segon autentifica l'usuari. Així doncs la funció principal de la paraula clau és evitar que un usuari es faci passar per un altre. A aquest efecte, cal mantenir la paraula clau sempre en secret.

Algunes vegades, quan no és perillós que un usuari es faci passar per un altre, pot ser que la paraula clau sigui nul·la, és a dir, que no calgui usar paraula clau per començar la sessió.

A continuació, iniciarem una sessió de treball en el computador. A aquest efecte cal que demanem la paraula clau i el nom d'usuari amb què hem de treballar al professor de laboratori o a l'operador.

El pas següent és engegar el computador i la pantalla. Segons la configuració, és possible que engegant el computador la pantalla també s'engegui.

Quan la màquina s'engega apareix una caràtula amb un missatge que diu:

```
login:
```

Heu de respondre aquest missatge amb el nom d'usuari seguit de `Return`

Un cop conegut l'usuari que vol començar la sessió, el sistema us demanarà la vostra paraula clau:

```
password:
```

Heu de respondre aquest missatge amb el nom d'usuari seguit de `Return`. Observeu que la paraula clau no apareix mai a la pantalla.

Una vegada hem acabat amb la feina, cal tancar la sessió. Això cal fer-ho sempre, sense excepció. Per fer-ho s'empra la comanda `logout`, amb la qual indiquem al sistema que no volem fer més ús dels seus serveis i que tanquem la sessió.

Exercici 1 *Engegueu la màquina i observeu tot el que succeeix a la pantalla. Feu un login amb el nom d'usuari que us correspongui. Una vegada l'hàgiu fet, feu un logout.*

Exercici 2 *Quin és el significat del terme compte emprat anteriorment? Per quina raó s'usa aquest mot?*

2.3.3 El sistema operatiu

Una vegada heu fet el *login*, apareixen a la pantalla uns missatges indescriptibles i, a continuació, un text ajustat a l'esquerra. Aquest text té una forma diferent a cada ordinador, però generalment presenta un aspecte semblant a algun dáquests:

```
pep@desmai: ~/tmp$   /home/pep>
$                    >
```

Heu caigut a les mans del sistema operatiu! El sistema operatiu és un programa complex que controla tots els recursos del computador i ofereix a l'usuari una interfície apropiada per treballar amb la màquina. Al sistema operatiu se li dónen instruccions —comandes de sistema— escrivint-les darrera el *prompt*. El *prompt* és un missatge que escriu el sistema per indicar que està esperant una comanda. Sempre que a la vostra pantalla hi hagi el *prompt* (missatge del sistema) esteu sota el control del sistema operatiu. Més endavant estudiarem quin és el significat del *prompt*. Per a les pràctiques, i tenint en compte que a cada instal·lació el *prompt* pot ser diferent, sempre usarem el mateix prompt en els exemples: `$`. Feu les traduccions oportunes per a la vostra instal·lació particular.

L'entitat més important manipulada pel sistema operatiu és el fitxer. Un fitxer és una entitat que conté informació susceptible de ser usada pel computador i que s'emmagatzema físicament en una unitat perifèrica, generalment un disc. Tot fitxer té un conjunt d'atributs que el caracteritzen. El més important és el seu nom o identificador. Altres atributs importants són el mida en bytes, els permisos per accedir al fitxer o la data en què va fer-se la darrera modificació.

A la unitat central hi ha un dispositiu que serà d'importància vital durant la resta del curs. Aquest dispositiu permetrà conservar la feina que feu dia a dia en les vostres pràctiques. Té, normalment, forma de forat de bústia: del forat se'n diu unitat de disquet i del que es posa en el forat, disc —encara que té forma quadrada!. El disc és un suport magnètic on el computador és capaç d'emmagatzemar informació diversa. Com a tal supòrt magnètic requereix un respecte especial quant a apropar-lo a fonts de magnetisme, ja que en un tres i no res podríeu perdre'n el contingut. Evidentment no és pas l'única forma de perdre el contingut: el disc no es pot tocar amb els dits ni es pot aixafar.

Un disc pot contenir diversos fitxers. La llista dels noms i atributs dels fitxers continguts en una unitat determinada s'anomena directori.

Exercici 3 *Si podem dir que un fitxer és, quant a la funció comparable a un llibre, a què és comparable un directori? Pista: on es posen els llibres?*

El primer que cal aprendre és a mirar què hi ha en un directori. El directori amb el qual treballarem s'anomena directori arrel de l'usuari (*home directory*). Si és el primer cop que treballeu amb l'ordinador probablement no contindrà cap fitxer visible. En aquest cas, en crearem un amb la comanda

```
cp /etc/fstab llegeix.me
```

Per veure el contingut d'un directori, cal usar la comanda `ls`. Escriviu, per exemple,

```
ls
```

Tot seguit la màquina us contestarà amb una llista de noms: és el contingut del directori. De fet, us ha mostrat la llista de fitxers visibles. Un directori també conté fitxers que normalment no apareixen al llistat que us presenta la comanda `ls`. Els noms d'aquests fitxers sempre comencen pel caràcter punt ('.'). Podeu fer que apareguin llistats afegint el qualificador `a`. Un qualificador és una opció que modifica el comportament de la comanda que qualifica. Si feu

```
ls -a
```

obtindreu els mateixos fitxers que quan heu fet `ls` més alguns fitxers que comencen per punt. Una altre qualificador interessant de la comanda `ls` és `l`. Si proveu

```
ls -l
```

obtindreu un llistat en el qual per cada fitxer es mostren alguns del seus atributs a més de nom. El resultat d'aquesta comanda és una cosa semblant a

```
-rw-r--r--    1 pep      users         346 mar  3 12:46 llegeix.me
```

Cada fila del llistat correspon a un fitxer i té l'estructura següent. La primera columna indica els permisos d'accés associats al fitxer. La tercera columna indica l'usuari al què pertany el fitxer (el propietari del fitxer). La ciquena columna indica la mida en bytes del fitxer. Les columnes sisena, setena i vuitena corresponen a la data en la qual s'ha modificat el fitxer per darrera vegada. La darrera columna mostra el nom del fitxer.

La informació que un fitxer conté viu físicament en un disc. I els discs s'estructuren jeràrquicament en directoris. A cada instant de la sessió hi ha un directori que rep el nom de directori per defecte. Podem conèixer quin és el directori per defecte amb la comanda `pwd`.

Exercici 4 *Quin és el nom del directori per defecte en el vostre cas?*

Sovint, el que es vol és veure les característiques d'un únic fitxer en comptes de les de tots els del directori. En aquest cas, podem indicar a la comanda el fitxer que ens interessa. Per exemple podem executar

```
ls -l llegeix.me
```

Encara més. També es possible consultar els atributs d'un subconjunt dels fitxers d'un cert directori. Per exemple, suposeu que us interessa veure les caracterí stiques dels fitxers que acaben amb `.txt` —els fitxers que contenen text. En aquest cas podeu utilitzar la comanda següent:

```
ls *.txt
```

Aquesta comanda utiltza un nou concepte que serveix per designar un grup de fitxers. És el que s'anomena patró (*pattern*). Quan indiquem a la comanda `ls` quins fitxers volem veure, el caràcter `*` significa "qualsevol paraula". De manera que, de fet estem demananant informació de tots els fitxers que tenen per nom qualsevol paraula seguida de `.txt`. Practiqueu els exercicis següents per entendre millor el concepte.

Exercici 5 *Proveu les comandes següents i, analitzant-ne els resultats, arribeu a comprendre com s'usen els patrons.*

```
ls llegeix.*

ls l*

ls *.me
```

Exercici 6 *Quina comanda més simple fa el mateix que la comanda* `ls *`*?*

2.3.4 Formatat d'un disc

De vegades resulta convenient guardar alguns fitxers en un disquet ja sigui per tenir una còpia de seguret o bé per transllladar informació a una màquina que no està connectada a la xarxa. Tots els disquets nous s'han d'inicialitzar abans de poder-los usar. El procés d'inicialització s'anomena formatatge. El formatatge d'un disquet solament es fa una vegada, abans d'utilitzar-lo el primer cop, i cal anar amb compte ja que *destrueix la possible informació que tingui emmagatzemada*. Per formatar el disquet introduïu-lo en la unitat lectora a fons (pareu compte a introduir-lo cara amunt!) i tot seguit executeu la comanda de formatatge `mformat` fent

```
mformat a:
```

Observeu que cal indicar quina és la unitat de disquets on hi ha el disquet que es vol formatar. És molt important que aquesta unitat sigui la correcta per evitar problemes majors. En general, la unitat de disquets correspon a la lletra `a:`.Quan aquesta comanda s'acabi d'executar —després de mostrar per pantalla estranys missatges—, el disquet estarà a punt per ser utilitzat.

Recordeu que:

- Formatar un disquet comporta destruir tota la informació que conté.
- Abans de treballar amb un disquet nou cal formatar-lo.
- Una vegada hem formatat un disquet no és necessari formatar-lo mai més.

Ara ja teniu el disquet a punt de solfa. Comproveu que no conté cap fitxer executant l'ordre

```
mdir a:
```

Exercici 7 *Quant espai buit queda al disquet acabat de formatar? Expressa la resposta en bytes, en Kbytes i en Mbytes. Tingues en compte que* $1\ Mbyte = 2^{10}\ Kbyte$ *i que* $1\ Kbyte = 2^{10}\ byte$.

Acabeu la sessió fent *logout*.

Capítol 3

Manipulació de fitxers

3.1 Material necessari

- Computador de l'aula d'informàtica.

3.2 Objectius

- Aprendre a utilitzar les principals comandes per manipular fitxers.

3.3 Guió de treball

Inicieu una sessió de treball amb l'ordinador.

En aquesta sessió s'expliquen un conjunt de comandes que permeten manipular els fitxers emmagatzemats en un disc. Abans de començcar, cal definir una categoria de fitxers molt important: els *fitxers de text*. Un fitxer és de text quan l'única informació que conté és una seqüència de caràcters —això és, un text. Els fitxers de text són importants per la seva àmplia utilització en les aplicacions informàtiques.

Els fitxers de text no són els únics tipus de fitxers existents. Un sistema operatiu també pot, generalment, treballar amb *fitxers binaris*. Aquests són fitxers que contenen dades en el format intern del computador; per tant, no se'n pot veure el contingut llevat que es faci amb eines especials.

Recordeu que, durant la darrera sessió, vàreu copiar al vostre directori arrel un fitxer de text anomenat `llegeix.me`.

Exercici 8 *Verifiqueu que el fitxer* `llegeix.me` *encara és al disc, tal com s'ha dit.*

Aquest fitxer és un fitxer de text. Realment, un fitxer de text seria molt poc útil si no poguéssim veure la informació que conté. Si en un moment donat volem veure quin és el contingut del fitxer ho podem fer usant la comanda `cat` per exemple,

```
cat llegeix.me
```

Exercici 9 *Mireu d'esbrinar quin és el nombre màxim de caràcters que caben en un fitxer de text. Pista: aneu copiant la Gran Enciclopèdia Catalana en un fitxer fins que la màquina es queixi per excés de grandària....*

Probablement us ha agradat tant el fitxer `llegeix.me` que voldríeu tenir-ne una còpia idèntica amb el nom `segon.txt`. Per copiar un fitxer sobre un altre podeu utilitzar la comanda `cp`. Executeu, per exemple,

```
cp llegeix.me segon.txt
```

Exercici 10 *Comproveu que els fitxers* `llegeix.me` *i* `segon.txt` *són idèntics.*

La comanda `cat` no solament serveix per mostrar el contingut d'un fitxer. Un altre ús habitual és el d'encadenar fitxers, és a dir, produir com a resultat el contingut del primer fitxer seguit del contingut del segon. Proveu-ho fent

```
cat llegeix.me segon.txt
```

En general, la comanda `cat` pot encadenar el contingut d'un nombre arbitrari de fitxers.

Observeu que fins ara sols hem après a copiar fitxers i a veure què contenen. La quantitat d'informació que pot encabir una màquina té un límit i, en conseqüència cal poder esborrar fitxers quan deixin de ser necessaris. Per fer-ho, hi ha una comanda per esborrar fitxers. *La comanda que esborra fitxers és perillosa*: és molt fàcil esborrar fitxers que no es volien esborrar. Els fitxers esborrats no es poden recuperar mai més. Per fer algunes proves, dupliquem el fitxer `segon.txt` i verifiquem que s'ha duplicat fent

```
cp segon.txt primer.txt
```

```
ls
```

Fetes les advertències, vegem com es pot esborrar un fitxer. Posem per cas que volem esborrar el fitxer de nom `segon.txt`. Simplement cal usar la comanda `rm` fent

```
rm segon.txt
```

amb la qual cosa es perd tota la informació que contenia el fitxer i el seu nom desapareix del directori. Comprovem-ho fent

```
ls
```

Com la comanda `rm` és potencialment perillosa, hi ha la possibilitat d'indicar que es vol confirmar un a un l'esborrament de cada un dels fitxers. Això es pot indicar amb el qualificador `i`. Proveu-ho fent

```
rm -i llegeix.me
```

Sovint, especialment en fitxers llargs, és molt interessant tenir-los escrits sobre paper. La impressió d'un fitxer sobre un paper és el llistat i l'aparell que produeix llistats és la impressora. En les instal·lacions de computadors connectats en xarxa, sovint hi ha una única impressora compartida per tots els computadors. Aquesta impressora es troba situada en la sala d'operadors i són els mateixos operadors els que s'encarreguen de recollir els llistats i fer-los arribar als usuaris seguint alguna norma.

Per poder imprimir un llistat cal usar la comanda `lpr`. Suposem que volem imprimir el fitxer `primer.txt`. Solament cal fer

```
lpr primer.txt
```

Assabenteu-vos, si en el vostre cas s'aplica, de les hores en què els operadors treuen els llistats i del lloc on els deixen. Quan toqui, podreu passar a recollir el vostre. El llistat és una eina de treball important, sobretot per repassar la codificació d'un programa fora de l'aula. Tot i això, el paper és car i convé no fer-lo malbé innecessàriament.

Una altra de les manipulacions freqüents que es fan amb fitxers és el canvi de nom, és a dir, canviar el nom del fitxer sense modificar-ne el contingut. Per canviar el nom d'un fitxer s'utilitza la comanda `mv`. Per exemple, si voleu que el fitxer `primer.txt` passi a dir-se `regio7.txt`, solament heu d'executar

```
mv primer.txt regio7.txt
```

Exercici 11 *Quina seqüència de comandes usaríeu per dur a terme un canvi de nom sense emprar la comanda* `mv`*?*

Amb això donem la sessió per acabada. Ara solament falta fer el *logout* usant la comanda `logout`.

Capítol 4

Estructura de directoris

4.1 Material

- Computador de l'aula.

4.2 Objectius

- Comprendre els conceptes lligats a la jerarquia de directoris i aprendre a treure'n profit.
- Aprendre a usar les comandes relacionades amb el sistema de fitxers.
- Aprendre a usar comandes de manipulació de fitxers relacionades amb l'estructura de directoris.

4.3 Guió de treball

4.3.1 Estructura de directoris

Engegueu el computador i comenceu una sessió.

En els capítols anteriors hem vist que un fitxer s'emmagatzema en un disc. En un disc cap un gran nombre de fitxers. Molt sovint aquests fitxers són tants que es fa difícil treballar-hi. En aquests casos és convenient disposar d'un mecanisme que permeti organitzar els fitxers emmagatzemats en un disc. A aquest efecte es defineix el subdirectori. Un subdirectori no és res més que un directori amb un nom que està contingut dins un altre directori. Uff! Analitzem amb cura aquesta definició: un directori és una mena de "carpeta" contenidora de fitxers. Si té nom, podem dir que la carpeta està "etiquetada". Per tant, un subdirectori és una carpeta etiquetada que conté fitxers. Però aquest subdirectori està contingut dins d'un altre directori —o carpeta. Per tant, tenim una carpeta que conté fitxers i altres carpetes que a la vegada contenen fitxers i altres carpetes i així *ad infinitum*. Si fem gràfica aquesta idea, tenim un diagrama com el de la figura 4.1.

Si, en comptes de representar aquesta estructura usant un diagrama de carpetes, la representem mitjançant un graf on els arcs indiquen "contingut en", obtenim un diagrama com el de la figura 4.2. És a causa de la forma del graf que l'estructura de directoris s'anomena arbòria o jeràrquica.

L'estructura jeràrquica és molt útil per organitzar els fitxers. Vegem-ne algun exemple. En primer lloc, suposeu quecal emmagatzemar les dades referents al cens de les principals poblacions catalanes. Cada població té un fitxer amb el nom de la població que conté les dades. Una forma lògica d'organitzar aquests fitxers és disposar de tants directoris com regions té el Principat, i dins del directori de cada regió, un directori per a cada comarca. Finalment, dins del directori

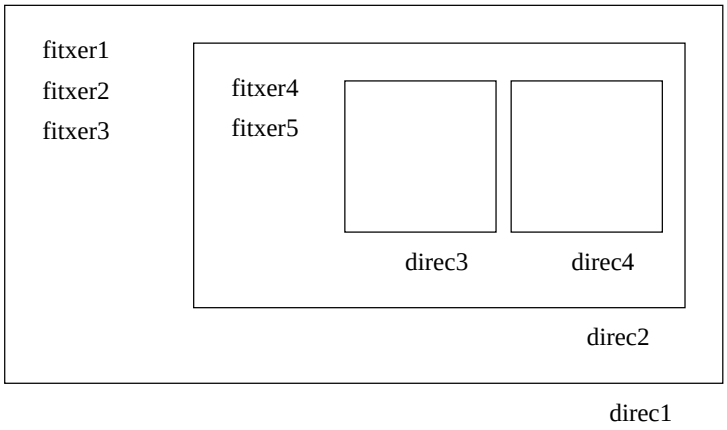

Figura 4.1: Directoris com carpetes

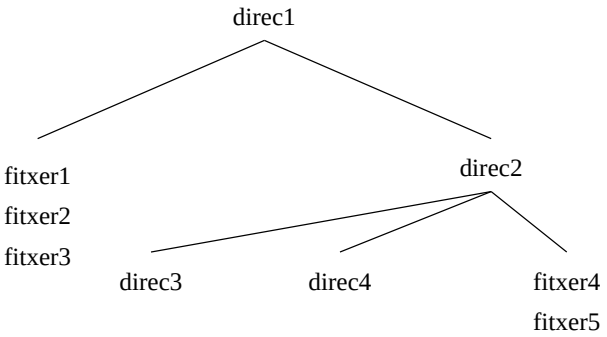

Figura 4.2: Arbre de directoris

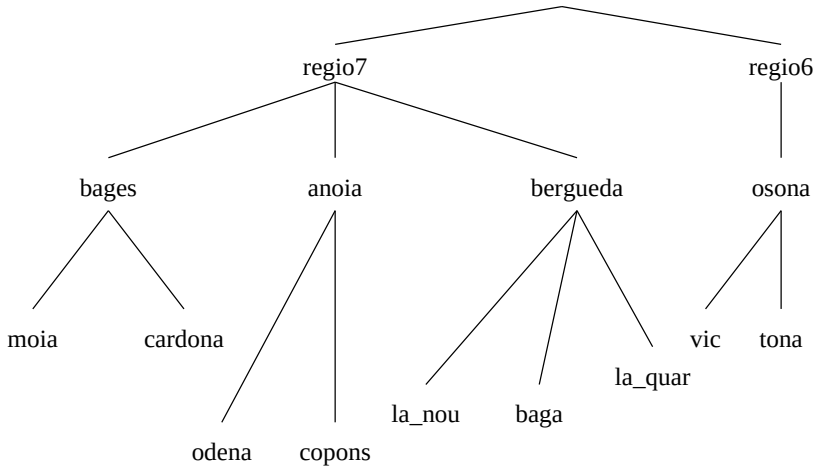

Figura 4.3: Organització en directoris de les dades de poblacions

de cada comarca hi hauria els fitxers corresponents a les principals poblacions de cada comarca. Això donaria com a resultat una estructura com la de la figura 4.3.

Un altre exemple clàssic és el cas en què un alumne d'una assignatura conserva la informació de diverses pràctiques. Li seria molt profitós mantenir un subdirectori per a cada pràctica i, dins d'un subdirectori, els fitxers referents a una pràctica en concret.

4.3.2 Creació i eliminació de subdirectoris

Una vegada vist quin és el concepte de directori cal conèixer les eines i els conceptes que permeten treure'n profit. Com a exemple suposarem que tenim un conjunt de fitxers que conserven informació referent a les eleccions municipals i que volem construir una estructura de directoris per organitzar-los.

Un cop iniciada la sessió, el directori per defecte és el vostre directori arrel.

Exercici 12 *Comproveu que el vostre directori arrel conté el fitxer* `regio7.txt`.

Anem a crear un subdirectori del directori per defecte. Tots els subdirectoris tenen un nom. Sigui `municip` el nom del subdirectori que anem a crear. El creem fent servir la comanda `mkdir` de la següent manera:

```
mkdir municip
```

Per comprovar que ha estat creat executem la comanda

```
ls
```

i observem que, a més del fitxer `regio7.txt`, hi ha el subdirectori `municip`.Podem mirar què és el que hi ha en el subdirectori `municip` usant la comanda `ls`

```
ls municip
```

Podem copiar un fitxer en el nou directori i comprovar que la còpia s'ha fet correctament mitjançant les comandes

```
cp regio7.txt municip/mataro.txt
```

```
ls municip
```

Encara podem fer coses més complicades. Per exemple, podem copiar en el subdirectori `municip` un fitxer amb nom `regio7.txt`. Simplement fem

```
cp regio7.txt municip
```

```
ls municip
```

La situació creada sembla estranya: hi ha dos fitxers amb el mateix nom. Com es poden distingir? La clau de volta es troba en el fet que cada un dels fitxers és en un directori diferent. En efecte, un directori no pot contenir fitxers amb el mateix nom però diferents directoris sí.

Suposem ara que volem conèixer els atributs del fitxer `regio7.txt` del directori per defecte i també els del fitxer `mataro.txt` del subdirectori `municip`. Per fer-ho, hem d'executar les comandes següents:

```
ls -l regio7.txt
```

```
ls -l municip/mataro.txt
```

Fixeu-vos que, en la segona comanda, hem indicat que el fitxer `mataro.txt` es troba en el subdirectori `municip` usant la sintaxi `municipi/mataro.txt`, és a dir, hem precedit el nom del fitxer amb informació que ens diu on es troba. En canvi, en la primera comanda no ha estat així.

Aquesta diferència s'explica en funció del concepte de directori per defecte. Per a cada unitat que hi ha en el sistema existeix un directori per defecte. Podem saber en cada moment quin és el directori per defecte usant la comanda `pwd`. Si executeu

```
pwd
```

veureu que escriu el quelcom semblant a `/home/pep`. Aquest és el directori per defecte en aquest moment.

Quan, com a paràmetre d'una comanda, escrivim un nom de fitxer sense indicar en quin directori és, sempre s'entén que aquest fitxer és al directori per defecte. En cas contrari, cal dir explícitament en quin directori és el fitxer.

El directori per defecte d'un disc es canvia usant la comanda `cd`. Per canviar el directori per defecte cap a `municip` cal executar

```
cd municip
```

Ara, si mireu el directori us trobareu els fitxers `regio7.txt` i `mataro.txt`. Per fer-ho, solament cal executar

```
ls
```

Fixeu-vos que no heu especificat el directori que volíeu veure i, en canvi, heu obtingut la informació. Això és natural, ja que ara el subdirectori `municip` és el directori per defecte. Normalment sempre s'usa com a directori per defecte aquell on hi ha els fitxers que s'utilitzen en cada moment. Així, si esteu treballant amb dades de municipis, per exemple, seria normal que el directori per defecte fos aquest.

Per tornar al directori arrel de l'usuari ho podeu fer amb la comanda `cd` sense arguments.

```
cd
```

Evidentment, un subdirectori també es pot esborrar. Per esborrar un subdirectori cal seguir els passos següents:

1. Esborrar tots els fitxers que conté.
2. Executar la comanda `rmdir`. Aquesta comanda sempre cal executar-la tenint en compte que no es pot esborrar el directori per defecte. En tot cas, primer cal canviar el directori per defecte.

Per veure un exemple d'esborrament, creeu un subdirectori amb alguns fitxers i després seguiu els passos per l'esborrar-lo. Per exemple, per crear-lo feu

```
mkdir proves

cp regio7.txt proves/f1.txt

cp regio7.txt proves/f2.txt

ls proves
```

Ara teniu un directori de nom `proves` que conté alguns fitxers. Per esborrar-lo cal seguir els passos explicats abans. Primer esborreu tots els fitxers que conté fent

```
rm proves/*
```

Tot seguit, com el directori que es vol esborrar no és el directori per defecte, procediu a esborrar-lo fent

```
rmdir proves
```

Exercici 13 *Pot haver-hi un subdirectori dins d'un subdirectori? Com s'hi treballaria? Proveu de crear un subdirectori dins d'un subdirectori, copieu-hi alguns fitxers i jugueu-hi (canvieu-los el nom, copieu-los, esborreu-los, . . .). Després esborreu tot allò que heu modificat.*

4.3.3 Camí d'accés a un fitxer

Fins ara hem vist què són els subdirectoris, com s'estructuren i com es poden crear o destruir. En la base d'aquests conceptes resta un punt important per explicar. Fixeu-vos en la comanda

```
ls municip/mataro.txt
```

En aquesta comanda, no solament s'indica el nom del fitxer que volem veure sinó també en quin directori se situa. Cal donar aquesta informació perquè el fitxer no és al directori per defecte. A l'exemple, es dóna la informació precendint el nom del fitxer amb el nom del directori seguit del caràcter `/` (`municip/`). Aquest text que precedeix el nom d'un fitxer i que diu en quin directori es troba s'anomena camí (*path*). Sempre que en una comanda cal anomenar un fitxer, es pot precedir el seu nom amb el camí que indica on es troba. Per exemple, proveu de fer

```
cat municip/mataro.txt
```

Hi ha dues maneres diferents de dir on es troba un fitxer o, d'altra manera, hi ha dues classes de camins:

Camí relatiu Forma de dir on és un fitxer que indica el camí que s'ha de seguir a través l'arbre de directoris, des del directori per defecte fins el directori on es troba.

Camí absolut Forma d'indicar on es troba un fitxer dient el camí que porta al directori des de l'arrel.

El directori arrel és l'únic directori que no és subdirectori de cap altre directori. Ens referirem a aquest directori amb el caràcter `/`. A partir del directori arrel podeu accedir a qualsevol fitxer o directori de la vostra màquina. Feu, per exemple

```
ls /
```

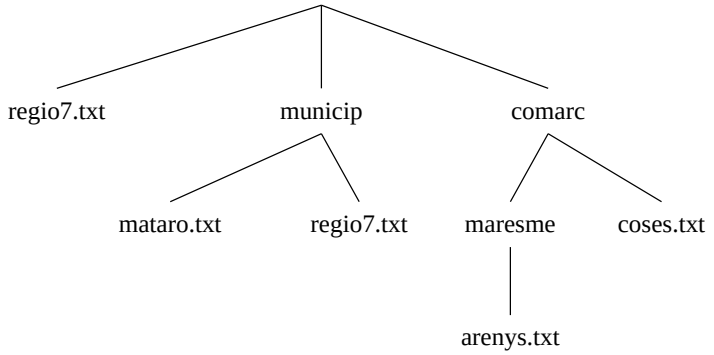

Figura 4.4: Estructura de directoris resultant

per veure quins fitxers i directoris conté el directori arrel. Cal no confondre el directori arrel amb el directori arrel de l'usuari. El directori arrel de l'usuari és un directori dins la jerarquia de fitxers a partir de qual un usuari pot crear fitxers i subdirectoris per tal d'emmagatzemar les seves dades. Ens referirem al directori arrel de l'usuari amb el caràcter `~`. Comproveu que el contingut del directori arrel de l'usuari no coincideix amb el directori arrel fent

```
ls ~
```

A continuació experimentarem amb camins relatius al directori per defecte i amb camins absoluts a partir del directori arrel de l'usuari. Per això, amplieu la jerarquia de directoris executant

```
mkdir comarc

mkdir comarc/maresme

cp regio7.txt comarc/coses.txt

cp regio7.txt comarc/maresme/arenys.txt
```

Després d'això, la vostra jerarquia queda esquematitzada a la figura 4.4.
Ara establiu com a nou directori per defecte el directori `comarc` fent

```
cd comarc
```

Suposeu, ara, que voleu veure el contingut del fitxer `mataro.txt`. El camí que va des del directori arrel de l'usuari fins el directori on és aquest fitxer és `~/municip/`. Noteu que sempre s'afegeix un caràcter `~` al principi del camí per indicar que és un camí absolut a partir del directori arrel de l'usuari. Amb això podem veure el contingut dient

```
cat ~/municip/mataro.txt
```

Fixeu-vos que hem expressat la localització del fitxer en qüestió usant un camí absolut a partir del directori arrel de l'usuari. Podríem fer el mateix usant un camí relatiu. En aquest cas cal anar del directori per defecte fins a l'arrel de l'usuari i després baixar fins el directori `municip`. Això ho escrivim així:
`../municip/`
Fixeu-vos en alguns detalls. Primerament, el camí no comença per titlla ni per barra perquè no hem donat un camí absolut. En segon lloc, observeu que, per indicar el camí d'un directori cap al seu superior, utilitzem dos punts. Per tant, per veure el contingut del fitxer `mataro.txt` expressant la seva posició mitjançant un camí relatiu cal fer

```
cat ../municip/mataro.txt
```

Els exemples que segueixen us ajudaran a entendre una mica millor el concepte de camí. Tots estan fets suposant que el directori per defecte és `~/comarc/maresme`. Estudieu-los amb cura sobre la figura 4.4, però no els executeu en el computador.

- Per esborrar el fitxer `mataro.txt` caldria fer

  ```
  rm ../../municip/mataro.txt
  ```

  o bé, usant el mode absolut,

  ```
  rm ~/municip/mataro.txt
  ```

- Per veure el fitxer `coses.txt` caldria fer

  ```
  cat ../coses.txt
  ```

  o bé, usant el mode absolut,

  ```
  cat ~/comarc/coses.txt
  ```

- Per copiar el fitxer `coses.txt` en el directori `municip` caldria fer

  ```
  cp ../coses.txt ~/municip/coses.txt
  ```

  Una altra possibilitat és

  ```
  cp ../coses.txt ../../municip/coses.txt
  ```

  També es pot dir, aprofitant el fet que volem copiar el fitxer amb el mateix nom,

  ```
  cp ../coses.txt ../../municip
  ```

  En aquest darrer cas, es pot ometre el nom del fitxer perquè la comanda sobreentén que serà el mateix.

4.3.4 Altres comandes de manipulació de fitxers

Tot seguit s'expliquen algunes comandes que actuen sobre fitxers però que tenen en compte la jerarquia de fitxers que els conté.

El qualificador `R` de la comanda `ls` permet llistar recursivament l'estructura de directoris. Proveu d'executar

```
ls -R
```

Si usem els qualificadors adequats, la comanda `cp` es pot fer servir per copiar subarbres de la jerarquia de directoris. El qualificador `R` permet copiar recursivament tots els subdirectoris d'un directori, a més dels fitxers. Executeu la comanda següent:

```
cp -R comarc comar2
```

Exercici 14 *Comproveu, usant la comanda `ls -R` que l'estructura de directoris actual és la que mostra la figura 4.3.4.*

Una altra de les comandes interessants és `mv`. Aquesta comanda permet moure fitxers o directoris dins la jerarquia. Per exemple, proveu d'executar

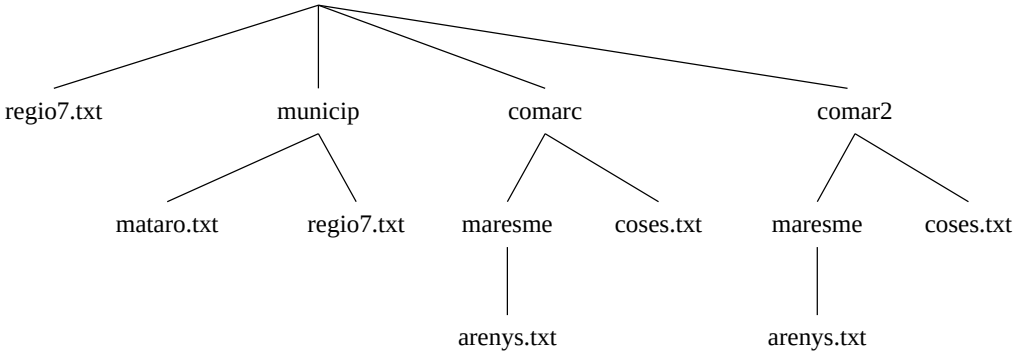

Figura 4.5: Estructura de directoris després de fer `cp -R`

```
mv regio7.txt regio.txt
```

El que ha passat és que el fitxer `regio7.txt` s'ha mogut cap a un altre fitxer: `regio.txt`. De fet, amb la comanda `move` no cal moure'l dins el mateix directori, també es pot moure cap a un directori diferent. Proveu de fer

```
mv regio.txt comarc
```

Comproveu que ara el fitxer `regio.txt` és en el directori `comarc`.

Exercici 15 *Quina diferència hi ha entre la comanda `mv` i la comanda `cp`?*

Exercici 16 *Moveu el fitxer `regio.txt` del lloc on es troba al directori arrel tot canviat-li el nom a `regio7.txt` i usant solament una única comanda `mv`. Comproveu que tot ha anat com s'esperava.*

La tercera utilitat de la comanda `mv` és la de canviar el nom de subdirectoris. Per exemple, podem canviar el nom del directori `comarc` i anomenar-lo `com` fent senzillament

```
mv comarc com
```

Exercici 17 *Comproveu que el nom del directori ha canviat tal com s'esperava.*

També és important conèixer el qualificador `R` de la comanda `rm`. Aquesta comanda permet no tan sols esborrar fitxers sinó subarbres complets de la jerarquia. Proveu d'executar

```
rm -R comar2
```

Després d'executar-se, el directori `comar2` i tots els fitxers i directoris que conté han desaparegut.

Exercici 18 *Dibuixeu la jerarquia de directoris que resulta d'executar la comanda `rm -R` anteriorment proposada.*

Amb aquest exemple acaba la sessió. No oblideu de fer *logout*.

Capítol 5

Redirecció i canals

5.1 Material

- Computador de l'aula.

5.2 Objectius

- Conèixer el concepte de redirecció i la seva utilitat.
- Conèixer el concepte de *pipe* o connector i la seva utilitat.
- Conèixer les principals comandes que poden actuar com a filtre i el concepte de filtre.
- Conèixer quins fitxers especials hi ha i quin ús se'ls dóna.

5.3 Guió de treball

5.3.1 Canals d'entrada i sortida. Redirecció

Les comandes o programes que s'executen des de l'intèrpret de comandes sovint escriuen resultats a la pantalla o llegeixen dades del teclat. Tot seguit estudiarem amb més deteniment els mecanismes relacionats amb aquest fet i com poden usar-se de maneres més interessants. Executeu la comanda següent:

```
ls -l
```

El resultat que se n'obté és un text o seqüència de caracters que són escrits a la pantalla. El que ha succeït en realitat queda resumit a la figura 5.3.1. La comanda `dir` no escriu el resultat directament a la pantalla sinó que l'escriu sobre una mena de "conducte" d'informació anomenat canal. Un programa sempre té disponibles dos canals predefinits: el canal estàndard de sortida i el canal estàndard d'entrada. El primer és el lloc on el programa escriu els resultats i el segon és el lloc d'on el programa llegeix les dades.

En principi, el canal de sortida sempre es troba connectat a la pantalla. D'aquesta manera, quan un programa escriu en el canal de sortida, la informació es visualitza a la pantalla automàticament. El canal actua com a mitjà de transport de la informació. L'avantatge de treballar amb aquest esquema és que el canal es pot redirigir cap a un dispositiu diferent d'aquell on habitualment està connectat. Proveu d'executar la comanda

```
ls -l > dades.txt
```

Exercici 19 *Quin efecte ha tingut l'execució d'aquesta comanda? Com l'explicarieu?*

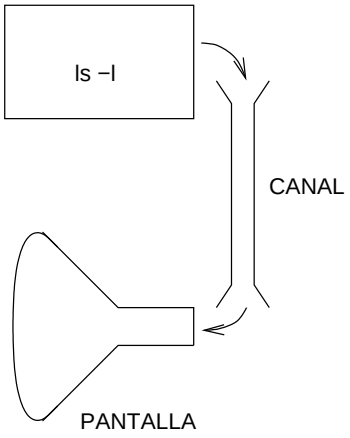

Figura 5.1: Esquema de funcionament de la comanda `ls`

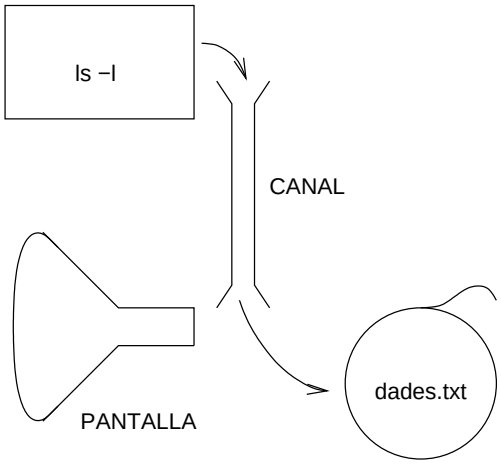

Figura 5.2: Esquema de funcionament de la redirecció de sortida

El que aquesta comanda ha fet és una redirecció del canal de sortida. El símbol `>` és l'utilitzat per indicar aquesta redirecció. L'efecte és molt senzill d'entendre: durant l'execució de la comanda `ls` el seu canal de sortida no s'ha connectat a la pantalla, com és habitual, sinó que s'ha connectat al fitxer `dades.txt`. El resultat és que tot allò que la comanda `dir` escriuria per pantalla ara és el contingut del fitxer `dades.txt`. La figura 5.2 mostra gràficament aquest procés.

Exercici 20 *Quin efecte produeix l'execució de la comanda següent?*
`ls > dades.txt`
Desapareix el contingut anterior del fitxer `dades.txt`*?*

Exercici 21 *Proveu d'executar ara la comanda*
`ls -l >> dades.txt`
Quina diferència de comportament s'observa respecte a l'exercici anterior? Què es pot dir de la redirecció `>>`*?*

Com haureu comprovat en el darrer exercici, la comanda `>>` també redirigeix el canal de sortida però l'efecte no és crear un nou fitxer sino afegir al fitxer que ja existia sense esborrar el contingut previ.

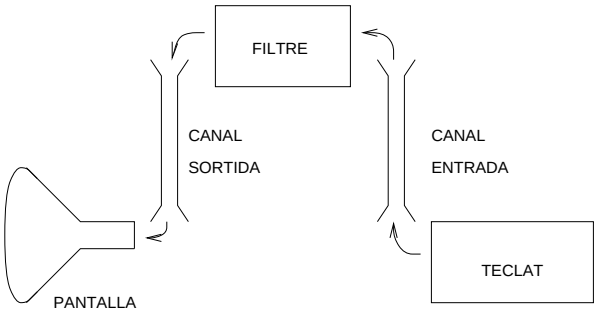

Figura 5.3: Esquema de funcionament d'un filtre

Exercici 22 *Què passa si es redirigeix el canal de sortida d'una comanda usant* `>>` *cap a un fitxer que no existeix ?*

Tot el que acabem de veure aplicat al canal de sortida també es pot aplicar al canal d'entrada. En aquest cas, la redirecció s'expressa usant l'operador `<` i l'efecte que es produeix consisteix a "desconnectar" el canal d'entrada del teclat i connectar-lo a un altre dispositiu. Més endavant apareixeran exemples de redirecció del canal d'entrada.

Exercici 23 *Feu un gràfic similar al de la figura 5.2 on s'expliqui el funcionament de la redirecció del canal d'entrada.*

5.3.2 Filtres

Els filtres són programes, habitualment força simples, que llegeixen dades del canal d'entrada, les processen d'alguna forma i escriuen els resultats en el canal de sortida, tal com es veu en la figura 5.3.

Executeu la comanda següent:

```
sort
```

i escriviu algunes línies de text com ara

```
El llegir fa perdre l'escriure
Qui gemega ja ha rebut
Abracadabra pota de cabra
```

Per acabar les línies de text escriviu una línia que solament contingui el caracter *final de fitxer*. En el nostre cas, aquest caràcter és el "control d" i s'obté polsant simultàniament la tecla de control (`Ctrl`) i el caràcter d. El resultat que la comanda escriu per pantalla és exactament

```
Abracadabra pota de cabra
El llegir fa perdre l'escriure
Qui gemega ja ha rebut
```

és a dir, les mateixes línies però ordenades alfabèticament. En efecte, la comanda `sort` és un filtre que llegeix línies i les torna a escriure ordenades alfabèticament. Naturalment, aquesta comanda es pot combinar amb la capacitat de redirecció. Així, si executeu la comanda

```
sort > ordenat.txt
```

i escriviu la llista dels noms dels vostres companys acabada en final de fitxer, obtindreu un fitxer `ordenat.txt` que conté la llista dels vostres companys ordenada alfabèticament.

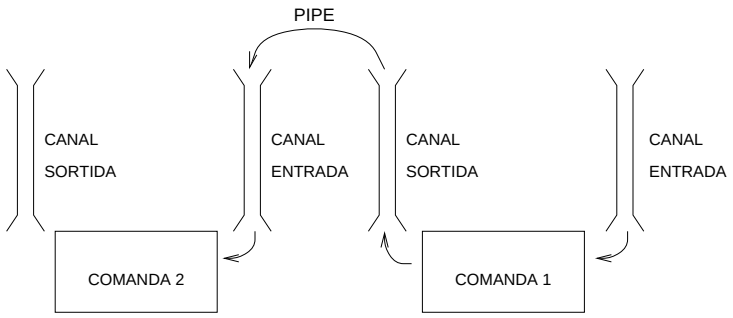

Figura 5.4: Esquema de funcionament d'una pipe

Exercici 24 *Comproveu el que s'acaba d'explicar.*

Un altre dels filtres importants és `grep`. Llegeix del canal d'entrada una seqüència de línies acabada en final de fitxer i escriu pel canal de sortida només aquelles línies que contenen un cert text. Proveu, per exemple, d'executar

```
grep garcia
```

i escriviu la seqüència de línies següent:

```
rodriguez patoso, javier
garcia capafonts, obdulia
del tomoso dieguez, dulcinea
pladellorens garcia, jaume andreu
```

Quin és el resultat d'aquesta execució?

5.3.3 Pipes

Una *pipe* és un element que permet connectar el canal de sortida d'una comanda amb el canal d'entrada d'una altra. D'aquesta manera, les dades de sortida de la primera comanda són també les dades d'entrada de la segona comanda. Podeu veure un esquema d'aquest mecanisme a la figura 5.4.

Un ús habitual de les *pipes* és connectar una comanda que genera dades amb un filtre que les post-processa. Per exemple, essent `x` un directori que contingui molts fitxers podeu provar d'executar la comanda següent:

```
ls -l | more
```

El resultat és que les dades escrites per `ls` són paginades per `more`.

Pipes i redireccions poden ser combinades entre elles per obtenir comandes més complexes. Per exemple, es pot escriure una comanda que, donat un cert fitxer `dades.txt` que conté una llista de noms i cognoms de persones, generi una llista ordenada de totes aquelles persones que es diuen `plademunt` i la deixi en el fitxer `pla-ord.txt`

```
cat dades.txt | grep plademunt | sort > pla-ord.txt
```

Exercici 25 *Escriviu i proveu una comanda o seqüència de comandes que genera la llista dels fitxers d'un directori que acaben amb* `.txt`*. La llista ha d'estar ordenada en ordre alfabètic invers.*

Les *pipes* són especialment importants per que permeten construir comandes complexes a partir de petites comandes molt simples. Aquesta filosofia trascendeix a les pròpies aplicacions escrites pels usuaris i permet que un usuari amplii el conjunt de comandes disponibles mitjançant comandes i programes escrits per ell mateix.

Capítol 6

Procés d'edició. L'editor EMACS

6.1 Material

- Computador de l'aula.

6.2 Objectius

- Comprendre què és un editor de textos i la diferència que hi ha amb un processador de textos.
- Aprendre a utilitzar un editor de textos.

6.3 Guió de treball

6.3.1 Què és un editor de textos

Un editor de textos és un programa per crear, modificar i visualitzar un fitxer de text, el qual s'usarà posteriorment. Els processadors de textos són programes que s'utilitzen per a la composició d'un text.

Una de les tasques que cal fer en el procés d'elaboració d'un programa és l'edició del programa font. El programa font és un fitxer de text que, un cop compilat i muntat, donarà lloc a un fitxer executable. Així doncs, una de les etapes del procés d'elaboració d'un programa passa per la utilització d'un editor de textos.

L'editor que utilitzarem en les pràctiques és l'editor `EMACS`.

L'editor `EMACS` és un editor de pantalla, és a dir, un editor que mostra, normalment, el text que s'està editant per pantalla i l'actualitza de forma automàtica a mesura que es teclegen comandes.

Hi ha tota una família d'editors `EMACS`. Tots els editors d'aquesta família tenen en comú la seva organització; a més, segueixen la mateixa filosofia d'ús. Aquests editors són distribuïts per la Free Software Foundation sota una llicència de programari lliure. El membre d'aquesta família que utilitzarem és `GNU-Emacs`. Aquest editor pot ser utilitzat en màquines amb diferents sistemes operatius com `MS-DOS`, `UNIX` o `VMS`.

6.3.2 Com començar a treballar amb EMACS

El primer que un usuari ha de saber fer amb un editor és cridar-lo. La manera de cridar l'editor `EMACS` és

```
emacs [opcions]
```

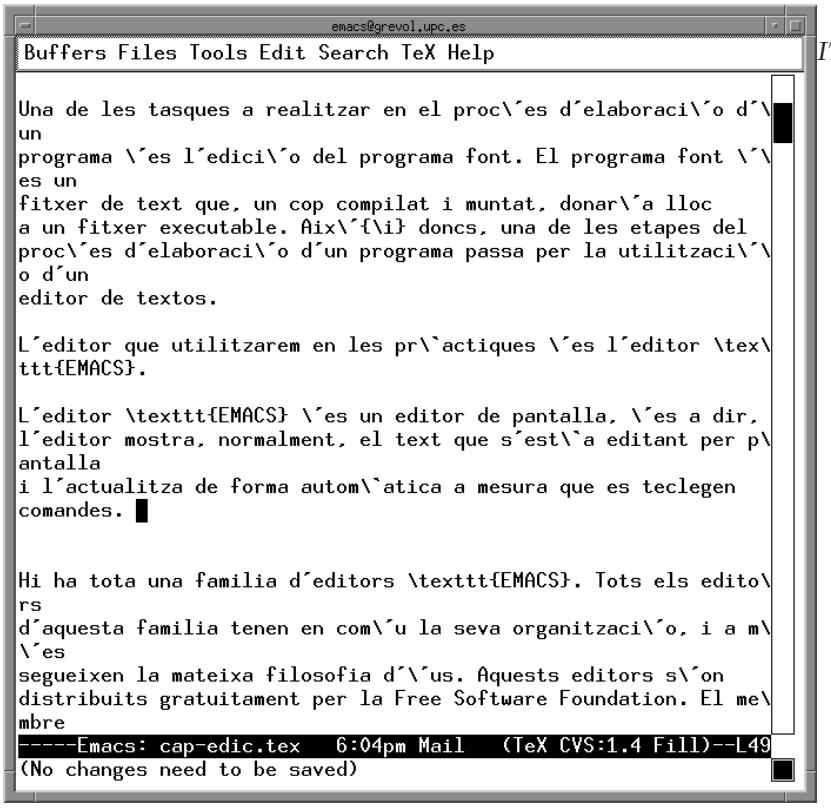

Figura 6.1: Imatge de la pantalla d'EMACS

Una opció [1] possible és el nom d'un fitxer; en aquest cas s'edita l'arxiu especificat. Les altres opcions corresponen a una utilització per a usuaris avançats. Si no utilitzem cap opció, obrim l'editor i ja podem començar a editar un text.

Un cop s'ha cridat l'editor `EMACS`, la pantalla es divideix en diferents àrees o finestres, com es veu a la figura 6.1.

L'àrea més gran és la que visualitza el text que s'està editant. Les altres regions contenen diferents tipus d'informació, que facilitarà la utilització de l'editor. La finestra en qué està situat el cursor s'anomena finestra seleccionada. En aquesta finestra és on les comandes d'edició tenen efecte.

Hi ha dues maneres diferents de sortir de l'editor `EMACS`, que s'executaran mitjançant comandes [2] diferents:

`<Ctrl> X <Ctrl> C`
: Serveix per matar el procés `EMACS`. Aquesta comanda ofereix la possibilitat de guardar les modificacions que s'han fet en el fitxer, si no s'han estat guardades, demanant confirmació abans de matar el procés.

`<Ctrl> Z`
: Suspèn `EMACS` de forma temporal i dóna l'oportunitat de reprendre'l posteriorment.

[1] Els parèntesis quadrats indiquen que l'usuari de l'editor té la possibilitat de no posar res més o bé posar una o més opcions entre tot un conjunt d'opcions possibles.

[2] La majoria de comandes que s'utilitzaran en editar amb l'editor `EMACS` utilitzen les tecles de control <Ctrl> o <Esc>. Quan utilitzem la notació <Ctrl> x, volem dir que hem de prémer la tecla Ctrl i mentre la tenim premuda, polsar la tecla x. En canvi, la notació <Esc> x, vol dir que hem de prémer la tecla <Esc>, deixar-la i tot seguit prémer la tecla x.

Exercici 26 *Una vegada hem vist com cridar l'editor, creeu un fitxer on volem escriure el text següent:*

```
El problema d'aproximar dades discretes (es a dir, punts
3D digitalitzats) amb un numero petit de superficies
polinomials a trossos suaus s'ha tractat en moltes arees diferents
com visio per computador, disseny assistit per computador i
enginyeria assistida per computador.

S'han desenvolupat diferents metodologies de treball per resoldre
aquest problema. Alguns sistemes utilitzen la metodologia anomenada
PCS (Punts - Corbes - Superficies). Un altre
metode per generar superficies a partir de conjunts de punts 3D es
l'anomenat PERS (Punts - Arestes (Edges) - Regions - Superficies).
```

Per editar aquest text, és a dir, per escriure'l i conservar-lo dins d'un fitxer, s'ha de triar el nom que donarem al fitxer. En aquest cas, li posarem el nom `reverse.txt`; per tant, cridarem l'editor fent

```
emacs reverse.txt
```

També podem cridar-lo sense el nom del fitxer, que ja posarem quan es guardi el fitxer

```
emacs
```

Ara ja podeu escriure el text com si utilitzéssiu una màquina d'escriure. En teclejar un caràcter ordinari imprimible, com la lletra 'a', s'insertarà en el document. Si es tecleja un caràcter de control com `<Ctrl>` o `<Esc>`, serà interpretat com l'inici d'una comanda d'una certa operació. En qualsevol moment es pot interrompre la comanda en curs prement `<Ctrl> G`, potser més d'un cop.

En escriure el text, si es vol saltar de línia haurem de premer la tecla `<CR>` i podrem continuar a la següent. Un cop s'ha escrit tot el text, podeu sortir amb `<Ctrl> X <Ctrl> C` i heu de respondre afirmativament quan es demani conformitat per guardar el fitxer. Hi ha altres comandes per guardar fitxers sense haver de sortir de l'editor, les quals es detallaran més endavant.

6.3.3 Comandes bàsiques d'edició

La utilitat d'un editor es deguda a la gran facilitat que proporciona per modificar el text d'un fitxer, sovint aprofitant el contingut d'altres fitxers. Aquest és el gran avantatge respecte a una màquina d'escriure.

Les principals comandes que s'utilitzaran per modificar un fitxer es descriuen en les següents subseccions, classificades segons la seva funcionalitat.

Moviments del cursor en el text

El cursor és aquella marca que indica on s'escriurà el caràcter següent. Les principals comandes de moviment del cursor són

`<Ctrl> F`
: Mou el cursor un caràcter a la dreta.

`<Ctrl> B`
: Mou el cursor un caràcter a l'esquerra.

`<Ctrl> N`
: Avança a la línia següent.

`<Ctrl> P`
: Mou el cursor a la línia anterior.

Aquests moviments del cursor també es poden aconseguir usant les tecles que trobareu a la part dreta del teclat. Assegureu-vos primer que el llum verd amb nom `BlocNum` és apagat; si no, apagueu-lo usant la tecla d'idèntic nom. `<Ctrl> F` és equivalent a la tecla →, `<Ctrl> B` és equivalent a ←, `<Ctrl> N` és equivalent a ↓ i `<Ctrl> P` és equivalent a ↑.

Altres comandes de moviment del cursor són

`<Ctrl> E`
: Avança el cursor al final de la línia en què està situat.

`<Ctrl> A`
: Situa el cursor a l'inici de la línia actual.

`<Esc> x goto-line`
: Permet moure el cursor a la línia n-èsima del text. `EMACS` sol·licita el valor de n.

`<Ctrl> V`
: Mou el text una pàgina cap a dalt. La pàgina que ara es visualitza és la següent de la que hi havia abans de fer aquesta comanda.

`<Esc> V`
: Mou el text una pàgina avall i es visualitza per pantalla la pàgina anterior.

`<Esc> <`
: Posa el cursor a l'inici del text.

`<Esc> >`
: Situa el cursor al final del text.

Exercici 27 *Practiqueu els moviments del cursor amb el fitxer reverse.txt. Situa't a la línia número 10 del fitxer.*

Comandes per esborrar, moure i copiar un text

Tota l'edició es realitza en un *buffer* (una regió de la memòria del computador) que emmagatzema el text. La manera usual d'editar és llegir un fitxer, col·locar-lo en el *buffer*, modificar el contingut del *buffer* i després escriure el *buffer* en el fitxer. `EMACS` crea *buffer*s com a àrea de treball temporal, on guarda text especialitzat com informació d'ajuda o parts d'un text que es voldran reutilitzar.

Hi ha dues maneres diferents d'esborrar un text: guardant-lo en un *buffer*, des d'on es podrà recuperar, o bé no guardant-lo. En aquest darrer cas, el text eliminat només es podrà recuprerar si es cancel·la l'eliminació amb la comanda d'eliminació que es veurà més endavant.

La manera usual de moure un text és esborrar-lo, guardar-lo en un *buffer* i després recuperar-lo un cop s'hagi mogut el cursor a la nova posició. Això també ens servirà per copiar un text. En aquest darrer cas, s'haurà de recuperar dues vegades: l'una immediatament després d'haver-lo esborrat i, l'altra, un cop situat el cursor en la nova posició. Tant el moviment com la còpia d'un text es poden fer dins d'un mateix fitxer o entre fitxers diferents, en aquest cas utilitzant les comandes d'operacions entre fitxers que s'exposen a continuació.

Les comandes per esborrar un text sense guardar-lo són

`<Ctrl> D`
: Esborra el caràcter que hi ha sobre el cursor sense guardar-lo.

`<Del>`
: Esborra el caràcter que hi ha a l'esquerra del cursor.

Les comandes d'esborrat següents guarden el text en un *buffer*:

`<Ctrl> W`
: Esborra el text d'una regió prèviament seleccionada.

`<Ctrl> K`
: Esborra el text des del caràcter que hi ha sobre el cursor fins al final de línia, però no esborra el salt de línia.

La recuperació d'un text esborrat mitjançant alguna de les comandes anteriors s'anomena *extracció*. L'extracció d'un text del *buffer* no en modifica la informació; per tant, és possible obtenir més d'una còpia del mateix text si s'extreu més d'un cop.

`<Ctrl> Y`
: Cópia, en la posició del cursor, el text que s'havia esborrat i guardat més recentment. El cursor quedarà situat al final del text copiat.

La manera de copiar un text, utilitzant les comandes vistes, ha de seguir els passos següents:

1. Esborrar amb `<Ctrl> W` (d'aquesta manera quedarà guardat en un *buffer*).
2. Reinserir el text en el lloc original amb `<Ctrl> Y`.
3. Moure el cursor a la posició on es vol la còpia i inserirà el text amb `<Ctrl> Y`.

Selecció d'un text

Hi ha moltes comandes d'`EMACS` que operen amb una part arbitrària d'un text (per exemple esborrar). Per especificar un text al qual s'aplicarà una d'aquests comandes, cal posar una marca en un extrem de la regió i el cursor en l'altre. Un cop hem definit una marca, aquesta no es modificarà fins que se'n redefineixi la posició.

`<Ctrl> <Espai>`
: Defineix una marca en la posició en què està situat el cursor.

Ara ja podeu practicar les comandes que acabem de veure. Per fer-ho utilitzeu el fitxer anteriorment creat amb nom `reverse.txt`.

Exercici 28 *Copieu el primer paràgraf del text* `reverse.txt` *al final del text. Mou el tros de text que està entre parèntesis al final del text.*

Cancel·lació de canvis

`EMACS` permet desfer tots els canvis que s'han fet en el text anul·lant les accions que s'hagin realitzat.

Les comandes que permeten anul·lar els canvis realitzats són

`<Ctrl> _`
: Anul·la la darrera comanda que s'ha executat.

`<Ctrl> x u`
: Anul·len el grup d'accions més recents que han modificat el text del *buffer* actual.

Si repetim les comandes anteriors, és possible anul·lar un nombre arbitrari d'accions. També es pot desfer una cancel·lació fent una acció senzilla i després cridant alguna de les comandes anteriors.

6.3.4 Operacions amb fitxers

Per editar un fitxer ja existent, l'editor l'incorpora al *buffer*. Això s'anomena *visitar* el fitxer. Les comandes que s'utilitzen des d'EMACS per visitar un fitxer són

`<Ctrl> X <Ctrl> F`
: Visita un fitxer i es visualitza per la finestra seleccionada. EMACS sol·licita el nom del fitxer.

`<Ctrl> X <Ctrl> R`
: Visita un fitxer en mode només de lectura i es visualitza per la finestra seleccionada. No s'hi poden fer canvis.

Observem que mitjançant la comanda `<Ctrl> X <Ctrl> F` podem tenir oberts més d'un fitxer. Cadascun estarà en un *buffer*, que tindrà el nom del fitxer corresponent.

Si volem guardar el contingut d'un *buffer* en un fitxer, hem d'utilitzar les comandes següents:

`<Ctrl> X <Ctrl> S`
: Guarda el contingut del *buffer* actual en un fitxer associat al *buffer*. Si no hi ha cap fitxer associat al *buffer*, EMACS demanarà el nom del fitxer.

`<Ctrl> X <Ctrl> W`
: Guarda el *buffer* actual en el fitxer especificat i associa el *buffer* a aquest fitxer a partir d'aquell moment. EMACS sol·licita el nom del nou fitxer.

Quan tenim més d'un fitxer obert, només se'n visualitza un a la finestra de treball. Si volem incorporar el contingut d'un fitxer obert o *visitat* a la finestra de treball, hem de fer la comanda

`<Ctrl> X b`
: Incorpora el *buffer* especificat a la finestra actual. EMACS demana el nom del *buffer*. Per defecte, incorporarà el *buffer* amb què s'havia treballat més recentment.

6.3.5 Cerca i substitució

Com altres editors, EMACS té comandes per fer cerques de cadenes de caràcters. La principal comanda de cerca és *incremental*, és a dir, comença a buscar abans d'haver acabat d'escriure la cadena de caràcters. També hi ha comandes de cerca no incremental, que no es detallaran ja que són més complicades d'utilitzar que les de cerca incremental.

Les comandes de cerca incremental més usuals són

`<Ctrl> S`
: Fa una cerca incremental cap endavant buscant la cadena de caràcters que hem introduït.

`<Ctrl> R`
: Fa una cerca incremental cap enrere buscant la cadena de caràcters introduïda.

EMACS situarà el cursor en el lloc on es troba la cadena que s'està buscant. Un cop hem fet una cerca d'una cadena de caràcters, podem buscar fàcilment noves ocurrències de la mateixa cadena de caràcters.

`<Ctrl> S`
: Buscar la següent ocurrència de la cadena de cerca que havia estat teclejada. Si teclegem `<Ctrl> S` després d'una cerca que no ha trobat la cadena de caràcters demanada, EMACS reiniciarà la cerca a partir de l'inici del *buffer*.

`<Ctrl> R`
: Buscar l'ocurrència de la cadena de caràcters prèvia.

L'editor avisarà si no ha trobat cap cadena idèntica a la que cercava.

Heu de tenir en compte que la cerca es farà sempre a partir de la posició del cursor (endavant o enrere, segons la comanda utilitzada).

Exercici 29 *Creeu un fitxer amb nom* `superf.txt` *que contingui el text següent:*

```
La construccio d'una superficie a partir de punts digitalitzats
es un problema que te moltes aplicacions, per exemple, grafics amb
computador, visio per computador, aplicacions CAD/CAM, entre altres.
Aquest problema de construccio es pot dividir en dos tipus:
aproximacio i interpolacio. En els metodes interpoladors, cada
punt dada es un punt de la superficie resultant.

En algunes aplicacions que treballen amb grans quantitats de dades, la
superficie que generen els metodes interpoladors requereixen
grans quantitats de memoria i la realitzacio de molts calculs. Un bon
metode, en aquesta situacio, es aproximar els punts donats amb una
superficie que definira un nombre de punts inferior.
```

Copieu el primer paràgraf del fitxer `reverse.txt` *al final del text* `superf.txt`*. Busque en el fitxer* `superf.txt` *la paraula superfície utilitzant les diferents comandes de cerca.*

6.3.6 Edició de programes

L'editor `EMACS` té un dispositiu que s'activa automàticament quan s'escriuen programes en un llenguatge de programació, en llenguatge `C` particularment, que en facilita l'edició. Entre les prestacions més importants que notareu en escriure programes font en llenguatge `C` hi ha la indentació automàtica de les estructures del llenguatge i la comprovació que hi ha un aparellament correcte de parèntesis de tota mena. Per activar aquest dispositiu, el nom del fitxer que es vol editar ha de tenir l'extensió `.c`, és a dir, ha de ser `nom_fitxer.c`.

`EMACS` té comandes per indentar correctament totes les línies d'un programa en `C`.

L'aparellament de parèntesis es realitzarà de forma automàtica en el text. En tancar un parèntesi, es mostrarà, bé a la finestra de treball, bé a l'àrea d'avisos (la línia de la part inferior, que s'utilitza per mostrar petites quantitats d'informació), on començava.

Si el parèntesi tancat no coincideix amb el darrerament obert -per exemple si s'escriu '[x)'- ens donarà un missatge d'avís a l'àrea d'avisos.

Exercici 30 *Tot seguit editarem un programa font en llenguatge* `C`*. Observeu com es fa l'aparellament dels parèntesis en editar.*

1. *Editeu, en un fitxer anomenat* `mitja.c`*, el programa font que hi ha a l'apèndix A.25.*
2. *Comproveu què passa si no tanqueu el parèntesi que hi ha a la capçalera de la funció* `LlegirEnter`*.*
3. *Feu una cerca de la paraula* `void` *i comptant el nombre de vegades que apareix.*

Capítol 7

Compilació, muntatge i execució d'un programa

7.1 Material

- Computador de l'aula.
- Compilador de C.
- Disquet de dades subministrat amb la documentació.

7.2 Objectius

- Conèixer els passos necessaris per generar un programa executable.
- Comprendre la funció bàsica del procés de compilació.
- Aprendre el funcionament elemental del compilador `gcc`.
- Comprendre la funció bàsica del procés de muntatge.
- Aprendre a muntar un programa executable.
- Aprendre a executar un programa.

7.3 Introducció

En el capítol anterior hem vist quin és el procés que s'ha de seguir per editar un programa en el computador i obtenir el que s'anomena *programa font*. Un programa font no és executable. A partir d'un programa font cal fer dues operacions més sobre el computador per obtenir un programa executable. Aquests dos passos són la compilació i el muntatge. En aquest capítol estudiarem què vol dir compilar un programa font i obtenir l'anomenat *programa objecte* i què s'ha de fer per compilar. Després veurem què cal fer per tal de muntar el programa amb l'objectiu d'aconseguir un programa executable.

7.4 El procés de compilació

El procés de compilació, és essencialment un procés de traducció. Es parteix d'un fitxer de text que no és més que un programa escrit en un determinat llenguatge i s'obté la traducció a un altre llenguatge. En el nostre cas, la informació inicial d'entrada del procés de compilació és el *programa*

font que s'ha obtingut com a resultat del procés d'edició. En acabar el procés de compilació, el que s'obté és el *programa objecte* o *unitat muntable* (figura 7.1), que és el programa font escrit en llenguatge màquina del computador en què estiguem treballant.

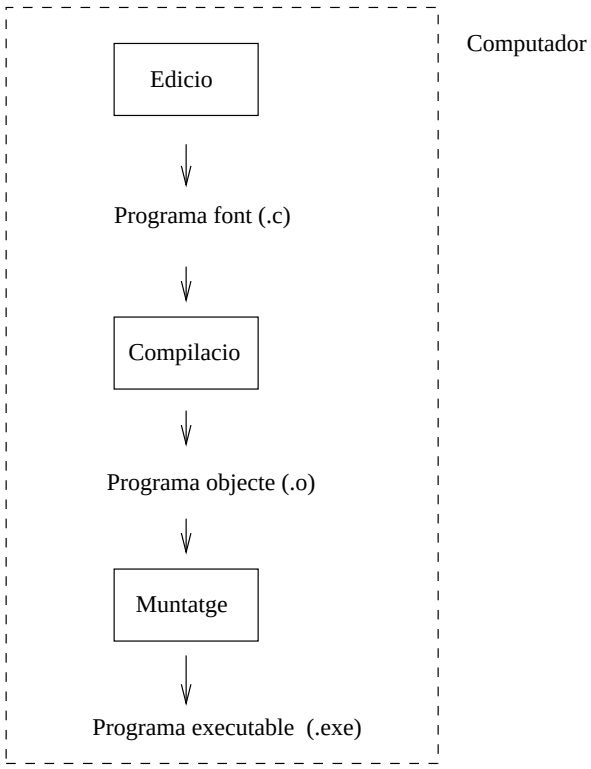

Figura 7.1: Etapes per generar un programa executable

L'eina que s'utilitza per fer el procés de compilació l'anomenem *compilador*. Un compilador és un programa que s'encarrega de fer la traducció d'un programa font a un programa objecte. Per a cada llenguatge de programació és necessari un compilador diferent. Per exemple, si es treballa en llenguatge C, és necessari un compilador de C, si es treballa en FORTRAN, cal un compilador de FORTRAN, etc. A més a més, dins de cada llenguatge de programació és possible trobar diferents compiladors d'un mateix llenguatge. Però, tot i que existeixen diferents compiladors, cal tenir en compte que majoritàriament tots tenen una part comuna que normalment compleix les especificacions fixades per a un estàndard. En el nostre cas, utilitzarem el compilador de C conegut amb el nom de `Gnu-C` i els nostres programes font s'ajustaran a l'estàndard `C-ANSI`.

7.5 Compilació d'un programa amb `gcc`

Per compilar un programa font escrit en C amb el compilador `gcc` i fer la compilació segons l'estàndard `C-ANSI` haurem d'escriure des de la línia de comandes del sistema operatiu

```
gcc -c [opcions] <nom_fitxer.c>
```

on `gcc -c` és la crida del compilador. Les opcions més usuals de compilació són, entre d'altres,

`-ansi`
És el *flag* de configuració del compilador perquè compili segons l'estàndard `C-ANSI`, encara que accepta certes variacions pròpies de `gcc`.

`-pedantic`
Aquest *flag* obliga el compilador a acceptar només programes font en `C-ANSI`, sense acceptar cap variació ni extensió de compilació. Aquest *flag* fa que el compilador és comporti de manera molt més estricta respecte a l'exigència del compliment de la norma `ANSI-C`.

`-O`
És el *flag* d'optimització. Permet detectar l'ús de variables no inicialitzades, reduir les dimensions del codi generat pel compilador i millorar el temps d'execució del programa.

`-Wall`
És el *flag* de configuració que indica al compilador que detecti totes aquelles situacions en el programa font que poden donar lloc a un *warning* o avís que hi ha alguna cosa que no s'està fent correctament del tot.

`-Wmissing-prototypes`
Amb aquest *flag* actiu, el compilador genera un missatge d'avís en el cas que hi hagi una funció o acció sense declaració de prototipus.

`-Wstrict-prototypes`
Amb aquest *flag* el compilador avisa en el cas que es defineixi o declari una acció o funció sense que s'especifiquin els prototipus.

Per tant, per compilar escriurem usualment les opcions següents:

```
-ansi -pedantic -O -Wall -Wmissing-prototypes -Wstrict-prototypes
```

A part dels *flags* del compilador presentats aquí, n'hi ha bastants més que permeten possibilitats de compilació que se surten de l'objectiu d'aquest llibre.

El compilador genera el programa objecte sobre un fitxer que per defecte es diu

```
nom_fitxer.o
```

Exercici 31 *Per tal de veure com es comporta la comanda de compilació, compileu com a prova el programa* `mitja.c` *(vegeu el llistat a A.25) i observeu què passa.*

Si tot va bé, no apareixerà cap missatge per la pantalla i es crearà el fitxer `mitja.o` amb el programa objecte generat pel compilador. Però si durant el procés de compilació es detecta que hi ha alguna incorrecció, mostrarà per pantalla un missatge de *warning* o un missatge d'error. Normalment, tant si es produeix un *warning* com un error, el compilador, a part del missatge, també diu en quina acció o funció s'ha produït la incorrecció.

Un *warning* no és el mateix que un error. Si es produeix algun *warning*, el compilador genera el programa objecte, però si hi ha algun error, el compilador no genera el programa objecte. Tot i que si hi ha *warnings* es pot continuar amb el procés de generació del programa executable, sempre és convenient eliminar-los tots, ja que a la curta o a la llarga un acaben produint un error en execució. Els missatges se'ns mostraran per pantalla amb el format següent:

```
nom_fitxer.c:numero_de_linia: warning: missatge_corresponent
```

on el compilador informa que en compilar el fitxer `nom_fitxer.c` a la línia indicada per `numero_de_linia`, hi ha alguna cosa que dóna lloc al missatge. Per exemple, si el compilador mostra

```
calcula.c:21: warning: unused variable ‘int count’
```

el compilador ens està informant sobre el fet que, en compilar el fitxer `calcula.c`, a la línia 21 hi ha declarada una variable que no es fa servir en cap lloc del programa. Això no és un error i el compilador genera igualment el programa objecte sobre el fitxer `calcula.o`, i, tot i que hi hagi el

warning, es podrà generar el programa executable; però si tenim una variable que no es fa servir no cal declarar-la.

Quan el compilador troba errors mostra un missatge per a cadascaun. Si hi ha errors, el procés de compilació no genera el programa objecte i, per tant, no es pot obtenir el programa executable. El format dels missatges d'error és el següent:

```
nom_fitxer.c:numero_de_linia: missatge_error_detectat
```

on el compilador està informant que, en compilar el fitxer `nom_fitxer.c`, a la línia indicada per `número_de_línia` s'ha trobat l'error que indica el missatge. Per exemple, si en compilar el fitxer `calcula.c` es mostra el missatge

```
calcula.c:52: parse error before ‘(’
```

se'ns està informant que a la línia 52 s'ha detectat un error de sintaxi. Cal tenir en compte que quan es detecta un error el compilador diu la línia en què es troba una situació d'error, però potser que l'error estigui just a la línia anterior.

Vegem tot seguit una sèrie de compilacions amb *warnings* i errors. Copieu en el vostre directori de treball els fitxers `graus1.c, graus2.c` (vegeu els llistats a A.11, A.12). Un cop copiats, compileu el fitxer `graus1.c`. Observeu que dóna els missatges de *warning* següents:

```
graus1.c: In function ‘main’:
graus1.c:29: warning: statement with no effect
graus1.c:22: warning: unused variable ‘k’
graus1.c:21: warning: ‘factor’ may be used uninitialized in this function
graus1.c: In function ‘LlegirReal’:
graus1.c:36: warning: unused variable ‘j’
graus1.c: At top level:
graus1.c:45: conflicting types for ‘EscriureReal’
graus1.c:16: previous declaration of ‘EscriureReal’
graus1.c: In function ‘EscriureReal’:
graus1.c:46: warning: double format, different type arg (arg 2)
```

S'està dient que:

- A la línia 29 es fa un càlcul que no es guarda enlloc i no es fa servir per a res. La línia 29 correspon a

  ```
  factor == PI/180.0;
  ```

  El que es vol fer en aquesta línia és assignar sobre `factor` la constant de conversió de graus a radians, però, com que hi ha l'error de no escriure correctament l'operador d'assignació (un sol = en C), el compilador interpreta que és l'operador de comparació =. No es genera un error de compilació sinó un *warning*. Amb aquest *warning* es pot continuar i arribar a obtenir el programa executable, però els resultats que donarà si s'executa seran totalment erronis.

- A les línies 22 i 36, hi ha dues variables declarades que no es fan servir enlloc i que, per tant, poden ser eliminades.

- A la línia 21 ha el càlcul d'una expressió en què es fa servir una variable que no està inicialitzada.

- A la línia 45 hi ha un tipus de variable que està en conflicte amb la declaració feta a la línia 16 de l'acció `EscriureReal`. Els tipus no es corresponen.

- A la línia 46 el format que s'indica en el `printf` no es correspon amb el tipus de variable.

Quan es compila un programa en C, és molt important revisar el programa font fins que pugui ser compilat sense *warnings* i s'obtingui un executable fiable. En l'exemple que estem estudiant, per fer desaparèixer aquests *warnings* cal esborrar de la línia 27 el = que sobra, eliminar les variables j, k del programa i a la línea 45, canviar el tipus *char* pel tipus *float*.

Ara compileu el fitxer `graus2.c` i observeu que dóna els *warnings* i missatges d'error següents:

```
graus2.c:16: warning: no previous prototype for ‘LlegirReal’
graus2.c: In function ‘LlegirReal’:
graus2.c:16: warning: function declaration isn’t a prototype
graus2.c:19: parse error before ‘{’
graus2.c:19: declaration for parameter ‘main’ but no such parameter
graus2.c:16: declaration for parameter ‘EscriureReal’ but no such parameter
graus2.c:19: number of arguments doesn’t match prototype
cc1: prototype declaration
graus2.c:29: parse error before ‘c’
graus2.c:31: warning: control reaches end of non-void function
graus2.c: At top level:
graus2.c:34: redefinition of ‘LlegirReal’
graus2.c:16: ‘LlegirReal’ previously defined here
graus2.c: At top level:
graus2.c:44: warning: no previous prototype for ‘EscriureReal’
```

Fixeu-vos que dóna tot un seguit de *warnings* relacionats amb el prototipus i dóna dos errors. Un error a la línia 19 i un altre a la línia 29. Tracteu primer d'eliminar el errors un a un, ja que aquests provoquen l'allau de *warnings*. Vegem, per exemple, el primer error. El compilador indica que ha detectat un error abans de la línia 19 que és el símbol { del programa principal. Si observeu les línies anteriors veureu que a la línia 15 falta un `';'`, que és el que provoca que tampoc es reconeguin les línies següents. Com les línies que no es reconeixen són les dels prototipus usats en aquest programa, apareixen *warnings* relacionats amb prototipus i paràmetres.

Esmeneu l'error de la línia 15 afegint un `';'` al final i torneu a compilar. Ara el misstages del compilador són

```
graus2-2.c:16: warning: function declaration isn’t a prototype
graus2-2.c: In function ‘main’:
graus2-2.c:29: parse error before ‘c’
graus2-2.c: At top level:
graus2-2.c:44: warning: no previous prototype for ‘EscriureReal’
graus2-2.c:44: conflicting types for ‘EscriureReal’
graus2-2.c:44: An argument type that has a default promotion
graus2-2.c:44: can’t match an empty parameter name list declaration.
graus2-2.c:16: previous declaration of ‘EscriureReal’
```

Observeu que encara hi ha *warnings* relacionats amb prototipus, però ja només a l'acció `EscriureReal`, i un error abans de la línia 29. Tractem primer d'eliminar l'error i després els *warnings*. Observeu que a la línia 28 falta escriure un `';'`. Si l'escriviu i torneu a compilar veureu que ja no hi ha cap error però que encara apareixen els *warnings* relacionats amb `EscriureReal`. Fixeu-vos en la declaració de prototipus i en la capçalera a la implementació de l'acció. És clar que són diferents i que a la línia del prototipus de l'acció `EscriureReal` hi falta escriure un `float` com a paràmetre formal. Per tant, la línia 16 és

```
void EscriureReal( float r );
```

7.6 Muntatge d'un programa

Un cop compilat un programa, el pas següent per arribar a obtenir un programa executable és el procés de muntatge. El procés de muntatge és el que s'encarrega d'enllaçar el nostre programa

objecte amb les altres unitats muntables que siguin necessàries per tal de generar un programa executable. Per exemple, si en el nostre programa hi ha una crida a una funció de càlcul de l'arrel quadrada `sqrt()`. La funció `sqrt()` és estàndard de la llibreria matemàtica de `C` i el programa objecte corresponent a la funció `sqrt()` és dins de la llibreria. En el nostre programa hi ha crides al càlcul de l'arrel i, per tant, quan es compili el programa es generaran crides a la funció, però el compilador, no afegirà el codi corresponent al càlcul de l'arrel. Quan es munti el programa, s'afegirà el programa objecte corresponent al càlcul de `sqrt()` i es resoldrà llavors la referència pendent.

Els altres programes objecte amb què es pot muntar el nostre programa objecte, tant poden haver estat generades per nosaltres com poden ser programes objecte de les llibreries subministrades amb el compilador de C, tal com veurem al capítol 9.

La comanda que s'ha d'escriure per muntar un programa objecte és

```
gcc <nom_fitxer.o> -o <nom_fitxer>
```

on s'està indicant que es vol muntar el fitxer `nom_fitxer.o` i amb el *flag* `-o` s'està indicant al compilador que el resultat del muntatge es vol sobre el fitxer `nom_fitxer`. El programa executable es genera sempre que no hi hagi errors en el muntatge. Normalment, el errors que es poden produir són deguts que hi ha alguna referència a una acció o funció de la qual no es té el programa objecte i que, per tant, no es pot muntar amb el nostre programa. Quan es produex un error de muntatge, el muntador dóna missatges de l'estil

```
undefined reference to 'nom_accio_o_funcio'
```

on s'indica quina és l'acció o funció que representa una referència no resolta.

El fet que no hi hagi el corresponent programa objecte d'una acció o funció provoca un error en el procés de muntatge però no en el procés de compilació. Per comprovar-ho, copieu en el vostre directori de treball el fitxer `graus3.c`. En compilar aquest fitxer no es genera cap *warning* ni error. Però, en muntar el programa objecte generat, el muntador genera un missatge d'error que hi ha una referència a `'EscriureReal'` que no està definida. Si observeu el llistat (vegeu la llista a A.12) veureu que l'acció `'EscriureReal'` no està implementada en el programa i el muntador no la troba a les llibreries que utilitza per generar el programa executable. Si al fitxer `graus3.c` afegiu la implementació de `'EscriureReal'`, l'error desapareix.

Si tenim un programa que fa crides a funcions de la llibreria matemàtica de `C`, quan es faci el muntatge del programa objecte caldrà indicar al muntador que tingui en compte la llibreria matemàtica. Per això, per muntar s'ha d'escriure

```
gcc <nom_fitxer.o> -lm -o <nom_fitxer>
```

on el *flag* `-lm` vol dir que volem muntar tenint en compte la llibreria matemàtica.

Exercici 32 *Copieu en el vostre directori de treball el fitxer* `graus.c`*. Compileu i munteu el programa per tal de generar el programa executable.*

Exercici 33 *Proveu de muntar el programa objecte del programa* `mitja.o`*. Comproveu que es genera el programa executable corresponent.*

7.7 Execució d'un programa

Un cop muntat un programa s'està en situació d'executar-lo. Per fer-ho només cal escriure el nom del fitxer executable en què s'ha guardat el resultat del muntatge.

Per executar el programa `graus.c` (vegeu el llistat a A.10) cal escriure

```
graus
```

Això fa que s'executi el programa *graus*, que transforma una quantitat donada en graus, minuts i segons en el seu equivalent en radians. Per introduir els valors inicials cal escriure tres valors reals pel teclat (canal estàndard d'entrada); el resultat de la conversió es veurà a la pantalla (canal estàndard de sortida).

Un cosa important que cal saber és com aturar un programa quan s'està executant. Si us trobeu en el cas que executeu un programa i, per exemple, hi ha una iteració infinita i per tant no acaba mai, polsant les tecles `<CTRL><C>` es provoca la parada o avortament de l'execució del programa.

Exercici 34 *Estudieu quines són les dades que necessita el programa* `mitja` *i executeu-lo entrant les dades directament pel teclat.*

Exercici 35 *Editeu un fitxer que es digui* `dades.dat` *i que contingui les dades que seran les d'entrada del programa* `mitja.c`*. Un cop fet això, executeu el programa* `mitja` *redireccionant l'entrada des del fitxer* `dades.dat`*.*

Capítol 8

Compilació separada

8.1 Material

- Computador de l'aula.
- Disquet de treball.
- Disquet de dades subministrat amb la documentació.

8.2 Objectius

- Consolidar els conceptes relacionats amb els processos de compilació i muntatge.
- Comprendre els conceptes relacionats amb la compilació separada.
- Aprendre a compilar separadament un conjunt d'unitats de compilació separada.
- Aprendre a muntar un conjunt d'unitats muntables.

8.3 Introducció a la compilació separada

Recordem que anomenem compilació al procés de traducció automàtica d'un programa —font— escrit en un llenguatge d'alt nivell al mateix programa —objecte— escrit en un llenguatge de baix nivell. Sabem també que la compilació és un procés complex i costós. Especialment si el comparem amb el procés de muntatge. Durant el procés de muntatge es genera el programa executable a partir del programa objecte i els subprogrames que usa i que formen part de la llibreria estàndard del llenguatge. El muntatge és, doncs, un procés de resolució de referències.

Caldrà tornar a compilar un programa font quan hi haguem fet alguna modificació. Les modificacions d'un programa es poden produir, entre d'altres raons, perquè durant la fase de prova s'ha detectat algun error o bé perquè se n'ha tornat a dissenyar alguna part —per millorar-ne l'eficiència, per exemple. Si el programa amb què estem treballant és gran, una modificació sobre una part petita implicarà tornar a compilar tot el programa. És a dir, que caldrà repetir un procés costós sobre tot el programa encara que la part que estrictament requereix tornar a ser compilada sigui ser molt petita.

El cost elevat de tornar a compilar un programa gran és una de les raons que motiven la seva fragmentació en unitats més petites, que seran compilades separadament. L'altra raó és que, quan s'aborda el disseny d'un programa complex, cal usar tècniques, com el disseny modular o amb tipus abstractes de dades, que complementin la tècnica de disseny descendent. Aquestes tècniques permeten descompondre el disseny en mòduls que acabaran essent les unitats que compondran el programa.

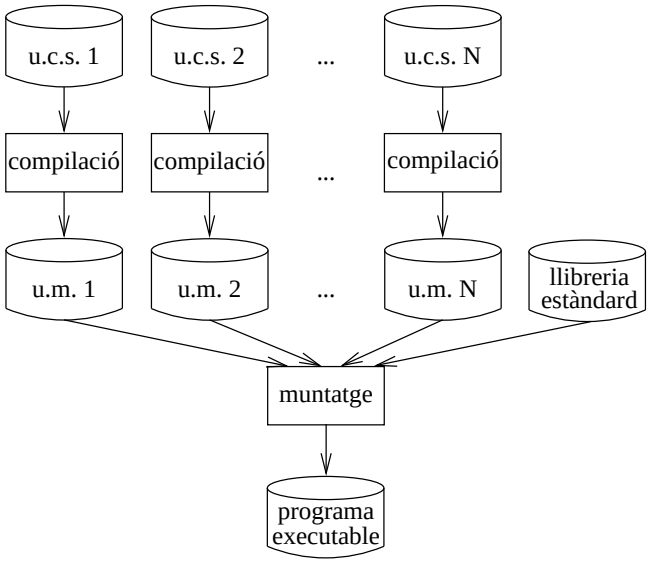

Figura 8.1: El procés de compilació separada

Direm que la *compilació separada* d'un programa és el procés de compilació aplicat separadament a cada una de les *unitats de compilació separada* en què s'ha descompost aquest programa. El resultat del procés de compilació separada és un conjunt d'unitats muntables. Una *unitat muntable* és el fragment de programa escrit en llenguatge de baix nivell que es correspon amb la traducció d'una unitat de compilació separada. Un dels avantatges de la compilació separada és que només cal tornar a compilar les unitats de compilació separada que s'han modificat.

El procés de muntatge, però, es complica lleugerament quan es treballa amb compilació separada. Per generar el programa executable cal partir de totes les unitats muntables, a més de la llibreria estàndard del llenguatge. Sempre que es torni a compilar una unitat de compilació separada s'haurà de repetir el procés de muntatge sobre totes les unitats muntables. La figura 8.1 mostra en un esquema el procés de compilació separada.

Els llenguatges de programació que suporten compilació separada han de definir quina és la sintaxi de les unitats de compilació separada. En el llenguatge `C` les unitats de compilació separada són els fitxers. Un programa en `C` es pot fragmentar en diferents fitxers `*.c`. La sintaxi d'un fitxer que conté un programa en `C` ens és coneguda, per tant no és necessari definir sintaxi nova per usar compilació separada.

8.4 Guió de treball

Suposeu que heu de resoldre el problema següent: donada una seqüència de números racionals pel canal estàndard d'entrada, calculeu-ne la suma. Un cop dissenyat l'algorisme, el programa en `C` corresponent es fragmenta en tres fitxers: `seqrac.c`, que conté el programa principal; `racional.c`, que conté la definició del tipus `Racional` i les seves operacions, i `natural.c`, que conté alguns subprogrames sobre els nombres naturals.

La comanda per compilar separadament cada fitxer `*.c` és

```
gcc -c -ansi -Wall -Wmissing-prototypes -Wstrict-prototypes fitxer.c
```

i la de muntage de més d'una unitat muntable és

```
gcc -o fitxer fitxer1.o fitxer2.o ... fitxerN.o
```

En primer lloc, compilareu separadament i muntareu els diferents fitxers que composen el programa. Per fer-ho:

1. Copieu els fitxers `seqrac.c`, `racional.c`, `racional.h`, `natural.c` i `natural.h` al vostre directori de treball.
2. Compileu separadament els fitxers `seqrac.c`, `racional.c` i `natural.c`.
3. Munteu les unitats muntables obtingudes en el punt anterior.
4. Experimenteu amb diferents seqüències de racionals acabades amb `0 1`. Els racionals es donen com parells d'enters `n d`, on `n` és el numerador i `d` el denominador.

Exercici 36 *Quin procés hauríeu de seguir per obtenir el programa executable si haguéssim modificat el contingut del fitxer `racional.c`?*

Haureu observat que, llevat del fitxer `seqrac.c`, que conté el programa principal, la resta d'unitats de compilació separada consten de dos fitxers: l'un amb extensió `*.c` i l'altre amb extensió `*.h`.

1. Estudieu el contingut dels fitxers `racional.h` i `racional.c`.
2. Compareu-ne l'estructura amb la d'un fitxer que contingui un programa complet en `C`.

Efectivament, els fitxers amb extensió `*.h` contenen les parts d'inclusions, definició de constants, definició de tipus i els prototipus dels subprogrames. Els fitxers amb extensió `*.c` contenen la implementació dels subprogrames. D'alguna manera, les unitats de compilació separada es divideixen en una *part d'especificació*, el fitxer `*.h`, i una *part d'implementació*, el fitxer `*.c`. En `C` només es compilen les parts d'implementació, és a dir, els fitxers que, per conveni, tenen extensió `.c`.

Exercici 37 *Quina estructura hauria de tenir un fitxer que contingués el programa complet que calcula la suma dels racionals d'una seqüència? Per compilar el fitxer resultant no ha de ser necessària la compilació separada.*

Centrem-nos en quin és el significat de les directives del preprocessador `#include` que apareixen en els diferents fitxers.

1. Anoteu per a cada un dels fitxers `*.c` i `*.h` quins fitxers inclou.
2. Dibuixeu un diagrama on cada fitxer es representi amb una caixa i on hi hagi una fletxa orientada de la caixa `A` cap a la caixa `B` si i només si el fitxer `A` inclou el fitxer `B`. Aquest diagrama s'anomena *diagrama d'usos o de dependències* entre fitxers.

El diagrama d'usos entre fitxers ens permet saber quin és el conjunt mínim de fitxers que hem de recompilar després de modificar-ne algun. Vegem-ne un exemple: suposem que el fitxer `natural.h` ha estat modificat. Observant el diagrama d'usos veiem que aquest fitxer és inclòs per `natural.c` i `racional.c`. Com aquests fitxers no són inclosos per cap altre, seran el únics que haurem de recompilar.

Exercici 38 *Quins fitxers caldrà recompilar si es modifica el fitxer `racional.h`? I si el modificat és `natural.c`? I si es modifiquen al mateix temps `natural.h` i `racional.h`?*

Finalment estudiarem quina és la utilitat de posar entre parèntesis el contingunt dels fitxers `*.h` entre les directives del preprocessador `#ifndef` i `#endif`. Amb aquest objectiu modificarem lleugerament l'enunciat del problema. En comptes d'imprimir la suma de la seqüència de nombres racionals volem imprimir la seva fracció canònica corresponent. Per això hem decidit implementar la funció `FraccioCanonica` en el fitxer `fc.c`. El fitxer `fc.h` contindrà la part d'especificació. També haurem de modificar el fitxer `seqrac.c` de manera que inclogui `fc.h` i, en comptes d'escriure la suma, haurem d'escriure el valor de `FraccioCanonica(suma)`.

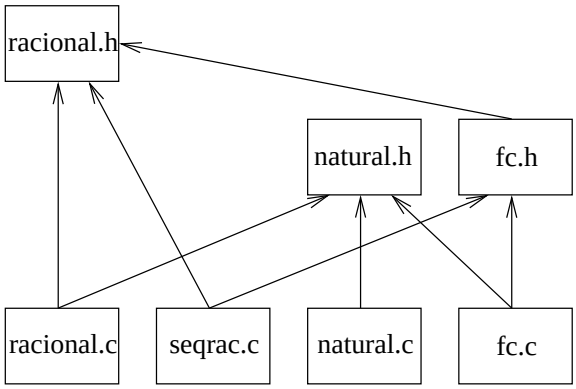

Figura 8.2: Diagrama d'usos entre fitxers

1. Copieu els fitxers `fc.c` i `fc.h` al vostre directori de treball.
2. Modifiqueu el fitxers `seqrac.c` afegint la línia `#include "fc.h"` després de la línia `#include "racional.h"`. Substituïu també la línia `EscriureRacional(suma);` per la línia `EscriureRacional(FraccioCanonica(suma));`.
3. Modifiqueu el diagrama d'usos entre fitxers afegint-hi els nous fitxers i les noves dependències. (Vegeu la figura 8.2).
4. Compileu separadament els fitxers `*.c` que ho requereixin i munteu-los.
5. Experimenteu amb diferents seqüències de racionals.

Com haureu observat, ni la compilació ni el muntatge han produït errors i el programa calcula el resultat esperat. Anem a fer alguns canvis que ens permetran constatar la importància dels parèntesis `#ifndef` i `#endif` en els fitxers `*.h`:

1. Esborreu la primera, la segona i la darrera línia dels fitxers `racional.h` i `fc.h`, que comencen per `#ifndef`, `#define` i `#endif`, respectivament.
2. Compileu el fitxer `seqrac.c` i observeu el resultat de la compilació.

El compilador generarà un error de redefinició de tipus. Aquest error es produeix quan es torna a definir un tipus que ja s'havia definit prèviament. Estudiem el que ha passat. Observant el diagrama d'usos entre fitxers veiem que en el fitxer `seqrac.c`, d'una banda, s'inclou el fitxer `racional.h`, que conté la definició del tipus racional. D'altra banda, s'inclou el fitxer `fc.h`, que al mateix temps inclou el fitxer `racional.h`. Aquest és, doncs, el problema. En `seqrac.c` s'inclou, per dues vies diferents, el fitxer `racional.h` i per tant totes les definicions que conté ocorren dues vegades. Els parèntesis `#ifndef` i `#endif` garanteixen que un mateix fitxer només s'inclougui una vegada.

Exercici 39 *Estudieu les directives del preprocessador* `#ifndef`, `#define` *i* `#endif` *i esbrineu perquè els parèntesis aplicats en els fitxers* `*.h` *garanteix que aquests s'incloguin una única vegada.*

A partir d'aquest punt introduirem les nocions bàsiques per entendre el funcionament del programa `make`. El programa `make` automatitza els processos de compilació i muntatge d'un programa a partir d'una descripció del diagrama de dependències mitjançant un text. Anem a veure, en primer lloc, com codificar el diagrama de dependències mitjançant un text.

Repassem què succeeix en compilar un fitxer font, per exemple `fc.c`, per obtenir una unitat muntable `fc.o`. En la compilació, a més del fitxer `fc.c` hi han intervingut tots els fitxers `*.h`

```
fc.o: fc.c fc.h natural.h racional.h
natural.o: natural.c natural.h
seqrac.o: seqrac.c fc.h racional.h
racional.o: racional.c racional.h natural.h
```

Figura 8.3: Descripció textual del diagrama de dependències de la figura 8.2.

```
seqrac: seqrac.o racional.o natural.o fc.o
```

Figura 8.4: Dependències per al muntatge.

que aquest incloïa, més els inclosos per aquests i així successivament. Observant el diagrama de dependències de la figura 8.2 veiem que `fc.c` inclou `natural.h` i `fc.h`, i aquest darrer inclou `racional.h`. Escrivim aquesta informació en una línia i tindrem el següent:

```
fc.o: fc.c fc.h natural.h racional.h
```

Hem de llegir la línia anterior com segueix: per tal d'obtenir `fc.o` cal compilar `fc.c` i en la compilació també hi intervindran els fitxers `fc.h`, `natural.h` i `racional.h`. Direm que `fc.o` és *l'objectiu* mentre que la resta de fitxers, els de la dreta dels dos punts, són les *dependències*. Penseu que a partir d'aquesta informació podríem construir un programa que decidís si cal compilar o no `fc.c`. Aquest programa consultaria la data de modificació del fitxer `fc.o` i la compararia amb la data de modificació de cada un dels fitxers `fc.c`, `fc.h`, `natural.h` i `racional.h`. Si algun d'aquests fitxers s'hagués modificat posteriorment que `fc.o`, caldria tornar a compilar. Doncs, aquesta és justament la feina que fa `make`.

Fins ara només hem descrit les dependències de `fc.o`, però això cal fer-ho per tots els fitxers que formen part del nostre programa. Seguint el mateix procediment que abans, a partir del diagrama de dependències de la figura 8.2, obtindrem el resultat que mostra la figura 8.3.

Els objectius que apareixen a la figura 8.3 són sempre unitats muntables i les dependències són fitxers font. Ara bé, podem utilitzar la mateixa estratègia per descriure les dependències d'un programa executable. En aquests cas, l'objectiu és el programa executable, diguem-ne `seqrac`, i les dependències són totes les unitats muntables que formen el programa, és a dir `seqrac.o`, `racional.o`, `natural.o` i `fc.o`. Si escrivim com abans aquesta informació obtindrem la línia que mostra la figura 8.4.

En aquest moment ja tenim tota la informació que necessita el programa `make` per compilar i muntar el nostre programa. Aquesta informació espera trobar-la en un fitxer anomenat `Makefile`. Per tal d'experimentar amb el programa `make` cal que feu el següent:

1. Editeu un nou fitxer anomenat `Makefile` i copieu primer la línia de la figura 8.4 seguida de les línies de la figura 8.3.

2. En una finestra de terminal, canvieu al vostre directori de treball, esborreu totes les unitats muntables i el fitxer executable fent

   ```
   rm *.o seqrac
   ```

   executeu la comanda

   ```
   make
   ```

   i comproveu, pels missatges que escriu al terminal, que ha compilat i muntat el programa `seqrac`. Executeu-lo per comprovar que funciona com abans.

Fixeu-vos, però que `make` ha compilat usant una comanda diferent que la usual, `cc` en comptes de `gcc`, i que no hi ha posat les opcions de compilació que usem sempre. Ara veurem com configurar el fitxer `Makefile` per tal que `make` usi el compilador i les opcions que volem.

1. Editeu el fitxer `Makefile` i afegiu al començament del fitxer les dues línies següents:

```
CC=gcc
CFLAGS=-Wall -Wstrict-prototypes -Wmissing-prototypes -ansi
```

2. Torneu a esborrar les unitats muntables i l'executable i compileu amb `make`.

Observeu que ara les comandes de compilació que ha usat `make` coincideixen amb les que usem habitualment. La variable `CC` conté el nom del compilador de `C` que usarà `make` per compilar i la variable `CFLAGS` conté les opcions de compilació. Ambdues variables les podem definir com més ens convingui seguint la sintaxi de l'exemple anterior.

Bé, un cop tenim el fitxer `Makefile` adequat per aquest programa experimentarem amb els avantatges que suposa usar `make` en comptes de compilar manualment. Per això utilitzarem la comanda `touch` que canvia la data de modificació del fitxer que rep com a argument. És a dir, `touch` produeix el mateix efecte que obrir el fitxer amb l'`emacs`, fer-hi una modificació, desfer-la i desar el fitxer: el contingut del fitxer segueix essent el mateix que abans, però al sistema operatiu li consta que ha estat modificat.

1. Modifiqueu la data d'un dels fitxers font, per exemple `fc.h`, fent

```
touch fc.h
```

2. Executeu `make` per veure quins són els fitxers que cal recompilar quan es modifica `fc.h`.

3. Comproveu que són efectivament els que hauríeu calculat vosaltres manualment a partir del diagrama de dependències de la figura 8.2.

Exercici 40 *Repetiu l'anterior, modificant la data dels altres fitxers font del projecte d'un en un i executant `make` cada vegada.*

L'ús de `make` que hem vist aquí és el més simple possible. En general, `make` es pot usar per qualsevol tasca que impliqui dependències entre fitxers ja que és possible especificar quines són les comandes que cal executar per tal de resoldre les dependències. Per fer-vos una idea dels possibles usos i de la complexitat d'aquesta eina resulta instructiu fer un cop d'ull al manual, escribint, en una finestra de terminal, la comanda

```
info make
```

Capítol 9

Llibreries

9.1 Material

- Computador de l'aula.
- Disquet de treball.
- Disquet de dades subministrat amb la documentació.

9.2 Objectius

- Consolidar els conceptes relacionats amb la compilació separada i el muntatge.
- Comprendre els conceptes relacionats amb les llibreries.
- Aprendre les comandes de creació de llibreries, inserció, esborrament i consulta d'unitats muntables en una llibreria.
- Aprendre a muntar un conjunt d'unitats muntables i llibreries.

9.3 Introducció a les llibreries

La majoria dels programes fan ús d'un conjunt de subprogrames bàsics. Les funcions matemàtiques no elementals com sinus, exponencial o valor absolut, o les operacions d'entrada/sortida sobre els tipus elementals —caràcter, real, enter— en són alguns exemples. Per tal d'estalviar temps i esforçcos als programadors, tot llenguatge de programació ofereix un conjunt de subprogrames estàndard ja dissenyats i implementats. Aquests subprogrames s'acostumen a organitzar en un conjunt d'unitats de compilació separada. Cada una d'aquestes unitats agrupa subprogrames amb una certa afinitat. Per exemple, es podrien agrupar totes les funcions matemàtiques sobre els nombres reals en una unitat de compilació separada i les operacions d'entrada/sortida en una altra. Com que aquestes unitats de compilació separada serien usades per molts programes es podrien oferir ja compilades, la qual cosa permetria estalviar la fase de compilació als programes que les fessin servir. El problema és que normalment s'ofereixen molts subprogrames que s'agrupen en moltes unitats de compilació separada, amb la qual cosa haver de recordar els noms de les unitats muntables corresponents resulta forçca difícil. Aquesta és una de les raons que motiva l'existència de les llibreries.

Una *llibreria* permet agrupar un conjunt d'unitats muntables sota un únic nom. Aquesta agrupació no seria de gaire utilitat si en el procés de muntatge s'hagués de partir únicament d'unitats muntables. Sortosament, els muntadors accepten també llibreries, a més d'unitats muntables.

Aquest fet complica lleugerament el procés de muntatge ja que el programa executable resultant estarà format per totes les unitats muntables que s'enumeren explícitament en la comanda de muntatge, més algunes de les unitats muntables contingudes a les llibreries que s'enumeren explícitament, més algunes de les unitats muntables de la llibreria estàndard. Quines de les unitats muntables de les llibreries formaran part del programa executable? Doncs només les unitats muntables que implementin algun subprograma cridat dins del programa.

Les llibreries s'han de poder gestionar: crear llibreries, inserir, esborrar, extreure o substituir unitats muntables de la llibreria i consultar quines unitats muntables conté una llibreria. La gestió de les llibreries es fa mitjançcant un conjunt de comandes que s'estudiaran més endavant.

La llibreria que agrupa les unitats muntables on estan implementats els subprogrames estàndard d'un llenguatge de programació s'anomena *llibreria estàndard* del llenguatge de programació. És freqüent que aquesta llibreria no s'esmenti explícitament en la comanda de muntatge. Això només és possible si el muntador no es crida directament sinó des d'una altra comanda que coneix quin és el llenguatge de programació amb què s'ha escrit el programa que s'ha de muntar.

9.4 Guió de treball

La comanda de muntatge pot incloure llibreries fent

```
gcc -o ftxr ftxr1.o ftxr2.o ...ftxrN.o llib1.a ...llibM.a
```

Les comandes de gestió de llibreries són les que hi ha a la taula 9.1.

`ar rcsv llib.a um1.o ...`	crea la llibreria *llib*.a que conté *um1*.o ...
`ar rsv llib.a um1.o ...`	insereix/substitueix *um1*.o ... de *llib*.a
`ar dsv llib.a um1.o ...`	esborra *um1*.o ... de *llib*.a
`ar xsv llib.a um1.o ...`	extreu *um1*.o ... de *llib*.a
`ar tv llib.a`	llista el contingut de la llibreria *llib*.a

Taula 9.1: Taula de comandes més usuals per mantenir una llibreria.

El significat de cada una de les opcions de la comanda `ar` el teniu descrit a la taula 9.2.

`c`	Usada juntament amb l'opció `r`. Suprimeix el missatge d'avís que es produeix quan s'insereixen unitats muntables en una llibreria que no existeix.
`d`	Esborra unitats muntables de la llibreria.
`r`	Inseri, substituint-les si ja hi eren, unitats muntables a la llibreria.
`s`	Actualitza o crea un índex dels subprogrames continguts en cada una de les unitats muntables de la llibreria. El manteniment d'aquest índex permet millorar l'eficiència del procés de muntatge. Una manera alternativa d'actualitzar o crear aquest índex és usant la comanda `ranlib llib.a`.
`t`	Llista les unitats muntables contingudes a la llibreria.
`v`	Fa que es mostri informació sobre l'execució de la comanda.
`x`	Extreu les unitats muntables de la llibreria.

Taula 9.2: Opcions de la comanda `ar`.

La llibreria estàndard de `C` és normalment en un fitxer de nom `libc.a`. La llibreria estàndard té associats un conjunt de fitxers `*.h` que són les parts d'especificació associades a cada una de les unitats muntables que componen la llibreria. Tant la llibreria com els fitxers `*.h` són en dos directoris del sistema de fitxers. Tot i que hi ha una ubicació per defecte, el lloc on es trobin en el vostre sistema depèn de com s'ha instal·lat el compilador.

Exercici 41 *A partir de la descripció de la llibreria estàndard de `C` (vegeu l'apèndix A de [KERN88], per exemple), per quines unitats muntables creieu que està formada?*

Exercici 42 *Localitzeu el directori que conté la llibreria estàndard de `C` en el vostre sistema. Llisteu les unitats muntables que conté i compareu aquesta llista amb la que havíeu proposat a l'exercici 41.*

En tot aquest llibre s'ha usat un conjunt de subprogrames d'entrada/sortida —`LlegirCaracter`, `EscriureCaracter`, ... — amb dos objectius: proporcionar un conjunt d'operacions d'entrada/sortida independents del llenguatge de programació escollit i simplificar-ne l'ús fent abstracció de les particularitats dels subprogrames de la llibreria estàndard del llenguatge escollit. Aquests subprogrames s'usen en molts programes diferents i, per tant, és raonable crear una llibreria que els contingui. A continuació crearem una llibreria i en farem ús en el procés de muntatge tot treballant sobre l'exemple del capítol 8.

1. Copieu els fitxers `seqrac.c`, `boolea.h`, `racional.c`, `racional.h`, `natural.c` i `natural.h` al vostre directori de treball.

2. Copieu els fitxers `entsort.c` i `entsort.h`, que contenen, respectivament, la implementació i l'especificació dels subprogrames d'entrada/sortida sobre els tipus elementals: `EscriureCaracter`, `LlegirCaracter`, `EscriureEnter`, `LlegirEnter`, `EscriureReal` i `LlegirReal`.

3. Estudieu el contingut dels fitxers `entsort.c` i `entsort.h`.

4. Compileu separadament `entsort.c`.

```
gcc -c -ansi -Wall -Wmissing-prototypes -Wstrict-prototypes entsort.c
```

5. Creeu la llibreria `local.a` i incorporeu-hi la unitat muntable `entsort.o`.

```
ar rcsv local.a entsort.o
```

6. Editeu el fitxer `seqrac.c` per tal de suprimir els prototipus i la implementació de les operacions d'entrada/sortida de tipus elementals. Substituïu també `#include <stdio.h>` per `#include "entsort.h"`.

7. Compileu separadament els fitxers `seqrac.c`, `racional.c` i `natural.c`.

8. Munteu les unitats muntables obtingudes al punt anterior amb la llibreria `local.a`.

```
gcc -o seqrac seqrac.o racional.o natural.o local.a
```

9. Executeu el programa per a alguns jocs de prova.

Exercici 43 *Que passa si us oblideu d'esmentar la llibreria en el muntatge? I si oblideu alguna de les unitats muntables?*

Exercici 44 *Quina utilitat té crear una llibreria per contenir unitats muntables que només s'usaran en un programa?*

Suposeu que els subprogrames de `natural.c` també s'usen en diferents programes.

1. Inseriu la unitat muntable corresponent a la llibreria `local.a`.

2. Torneu a muntar el programa `seqrac`, aquest cop sense la unitat muntable `natural.o` que heu incorporat a la llibreria.

3. Executeu el programa per a alguns jocs de prova.

Exercici 45 *Modifiqueu algun dels subprogrames de* `entsort.c`*. Per exemple, feu que l'acció* `EscriureEnter` *escrigui els enters entre parèntesis. Compileu la nova unitat i substituïu-la en la llibreria* `local.a`*. Torneu a muntar el programa amb la nova versió de la llibreria. Executeu-lo per a alguns jocs de prova.*

Exercici 46 *Esborreu la unitat muntable* `natural.o` *de la llibreria* `local.a`*. Torneu a muntar i a executar el programa.*

Una manera habitual d'organitzar les llibreries pròpies és col·locant-les en un únic directori —per exemple, `~/llib`— accessible des de qualsevol directori de treball. De la mateixa manera, els fitxers `*.h` associats a cada una de les unitat muntables de les llibreries s'acostumen a col·locar en un únic directori — per exemple, `~/incl`. Els compiladors i muntadors ofereixen suport a aquesta organització mitjançcant opcions que permeten indicar directoris on el compilador cerca els fitxers `*.h` que no troba al directori actual i el muntador cerca les llibreries que no troba al directori actual. L'opció de compilació és `-I`*`directori`*. Hi ha dues maneres de muntar amb una llibreria que no es troba en el directori actual. La primera és donar el camí absolut on es troba la llibreria. La segona és usar l'opció `-L`*`directori`* del muntador. Aquesta opció té algunes particularitats: cal que el nom de la llibreria comenci per `lib` —per exemple, `liblocal.a`— i la llibreria s'ha d'esmentar usant l'opció `-l`*`nom`*, on *`nom`* és el nom del fitxer sense el prefix `lib` ni l'extensió `.a` —per exemple, `-llocal`.

1. Creeu un directori per a les llibreries. Copieu-hi la llibreria `local.a` amb nom `liblocal.a`.

```
mkdir meullib

cp local.a meullib/liblocal.a
```

2. Creeu un directori per als fitxers `*.h` i copieu-hi els fitxers `entsort.h` i `natural.h`.

```
mkdir meuincl

cp entsort.h meuincl

cp natural.h meuincl
```

3. Compileu i munteu el programa `seqrac` usant les opcions de compilació i muntatge adequades.

```
gcc -c -Imeuincl -Wall -Wmissing-prototypes -Wstrict-prototypes seqrac.c

gcc -c -Imeuincl -Wall -Wmissing-prototypes -Wstrict-prototypes racional.c

gcc -Lmeullib -o seqrac seqrac.o racional.o -llocal
```

4. Executeu el programa per a diferents jocs de prova.

Exercici 47 *Què passa si us oblideu l'opció* `-I` *en la compilació?*

Exercici 48 *Què passa si us oblideu l'opció* `-L` *en el muntatge? I si munteu sense l'opció* `-L`*, però substituiu* `-llocal` *per* `meullib/liblocal.a`*?*

Capítol 10

Memòria dinàmica

10.1 Material

- Computador de l'aula.
- Disquet de treball.
- Disquet de dades subministrat amb la documentació.

10.2 Objectius

- Comprendre els conceptes relacionats amb memòria dinàmica.
- Comprendre l'especificació de les operacions de reserva i alliberament de memòria dinàmica.
- Aprendre a usar les operacions de reserva i alliberament de memòria dinàmica dins de programes.

10.3 Introducció a la memòria dinàmica

La figura 10.1 mostra la distribució de la memòria d'un computador per a un programa en execució. El segment de codi conté el codi del programa en llenguatge màquina del computador. Un altre segment conté les variables globals i les constants del programa. Són objectes que tenen una adreçca fixa durant tota l'execució del programa. El segment que conté la pila d'execució és dinàmic. La part usada d'aquest segment va variant a mesura que progressa l'execució del programa. És en aquest segment on es reserva l'espai per a les variables locals i els paràmetres cada cop que es crida un subprograma. Aquest espai s'allibera un cop el subprograma ha estat processat. La resta de la memòria, el segment de *memòria dinàmica*, normalment no és usada pel programa.

En certes ocasions resulta convenient usar la memòria del segment de memòria dinàmica. Per això la majoria de llenguatges de programació ofereixen operacions de reserva i d'alliberament de memòria d'aquest segment. L'operació de reserva d'espai sol·licita un fragment de memòria suficient per encabir-hi un objecte d'un cert tipus i retorna un apuntador a l'espai de memòria reservat. La consulta i la modificació del valor emmagatzemat en aquest espai es fa mitjançcant aquest apuntador. L'operació d'alliberament d'espai rep un apuntador a un fragment de memòria prèviamanet reservat i l'allibera. Un error molt freqüent és el intentar consultar o modificar el valor emmagatzemat en l'espai de memòria associat a un apuntador quan aquest ha estat alliberat.

Figura 10.1: Model de memòria d'un programa en execució

10.4 Guió de treball

La reserva i l'alliberament de memòria en `C` es fa cridant els dos subprogrames de la llibreria estàndard de `C` `malloc` i `free`. Per tal d'usar-los dins d'un programa, cal incloure-hi el fitxer `stdlib.h`. Els prototipus es mostren a continuació:

```
void *malloc(size_t size);
void free(void *p);
```

La funció `malloc` retorna un apuntador a una regió de memòria prou gran per encabir un objete de dimensió `size`. Retorna el valor `NULL` en cas d'error. La dimensió d'un objecte de tipus `T` s'obté amb la directiva `sizeof(T)`. L'acció `free` allibera l'espai apuntat per `p`. Aquest espai s'ha d'haver reservat prèviament amb `malloc`. Si l'apuntador `p` conté el valor `NUL`, `free` no fa res.

Els programes `mdelem.c`, `mdtupla.c` i `mdtaula.c` ens serviran per exemplificar l'ús d'aquestes operacions.

1. Copieu els fitxers `mdelem.c`, `mdtupla.c`, `mdtaula.c` i `boolea.h` al vostre directori de treball.
2. El programa `mdelem.c` reserva i allibera memòria per a alguns objectes de tipus elemental. Estudieu com s'usen les operacions de reserva i alliberament de memòria i com accedeix mitjançcant apuntadors. Compileu, munteu i executeu el programa.
3. Repetiu per a `mdtupla.c` i `mdtaula.c` el que es demana al punt 2. El programa `mdtupla.c` reserva i allibera memòria per a alguns objectes de tipus tupla. El programa `mdtaula.c` reserva i allibera memòria per a alguns objectes de tipus taula.

Exercici 49 *Quines diferències hi ha en la reserva de memòria dinàmica per a objectes de tipus elemental, tupla i taula?*

10.4.1 Alliberament de memòria

Convé esmentar alguns errors freqüents relacionats amb l'alliberament de memòria. El primer d'aquests errors és produeix quan es reserva memòria però mai no se n'allibera. Evidentment, en algun moment exhaurirem tota la memòria disponible i la propera crida a reservar memòria produirà un error. Vegem a continuació un exemple d'aquesta situació:

1. Copieu al vostre directori de treball el fitxer `mdnoall.c`.
2. Estudieu-ne el contingut.
3. Compileu i munteu el programa.
4. Executeu-lo i observeu-ne el resultat.

5. Modifiqueu-lo perquè us informi de quants *Kbytes* de memòria dinàmica disposa el vostre computador.

Exercici 50 *Què passa amb la memòria que s'ha reservat, no s'ha alliberat, però que no hi ha cap apuntador dins del programa que hi apunti? A aquest fenomen en direm referències penjades.*

Un altre error freqüent consisteix a accedir a una regió de memòria que ha estat alliberada prèviament. Aquesta situació és equivalent a la consulta del valor d'una variable no inicialitzada. El valor que s'obté és indeterminat i, per tant, el comportament del programa és imprevisible.

1. Copieu el fitxer `mdaccall.c` al vostre directori de treball.
2. Estudieu-ne el contingut.
3. Compileu i munteu el programa.
4. Executeu-lo i observeu-ne el resultat.

Finalment, un error menys freqüent consisteix a alliberar una regió de memòria que prèviament no ha estat reservada. El comportament del programa en aquest cas també és imprevisible.

1. Copieu el fitxer `mdallnr.c` al vostre directori de treball.
2. Estudieu-ne el contingut.
3. Compileu i munteu el programa.
4. Executeu-lo i observeu-ne el resultat.

10.4.2 Sinònims

Un dels problemes relacionats amb l'ús d'apuntadors és l'aparició de sinònims —*aliasing*. Aquest fenomen es produeix quan és possible accedir a un mateix objecte —regió de memòria— de més d'una forma. Per exemple, en el següent fragment de programa `a` i `*ap` són sinònims. Estudiem-lo per veure quins problemes es deriven de l'existència de sinònims.

```
1:  int sinonims(void)
2:  {
3:      int a, *ap;
4:
5:      ap = &a;
6:      a = 2;
7:      *ap = 3;
8:      return a;
9:  }
```

Quin valor retorna la funció `sinonims`? Sembla raonable respondre que 2, però `a` i `*ap` són sinònims i la modificació del seu valor es pot fer mitjançcant `a` —com a la línia 6— o `*ap` —com a la línia 7. La resposta, per tant, és 3. A continuació us proposem experimentar amb aquest fenomen.

1. Copieu el fitxer `mdsino.c` al vostre directori de treball.
2. Estudieu-ne el contingut i deduïu quins seran els valors que s'escriuran.
3. Compileu i munteu el programa.
4. Executeu-lo i compareu el resultat amb el que havíeu deduït.

10.4.3 Exemple: vectors

El tipus `Vector` s'usa freqüentment en problemes de càlcul numèric. Normalment es representa mitjançcant una taula de reals:

const
  N : **enter** = 30
fconst
tipus
  Vector = **taula** [1..N] **de real**
ftipus

La dimensió del vector —`N`— es decideix en temps de disseny de l'algorisme. Això implica que si es vol treballar amb vectors d'una dimensió diferent, cal modificar el programa canviant el valor de la constant `N` al valor adient.

Una manera més flexible de treballar amb vectors seria definint el tipus `Vector` com

const
  $MaxDim$: **enter** = 50
fconst
tipus
  TauReal = **taula** [1..$MaxDim$] **de real**
  Vector = **tupla**
    dim : **enter**
    taula : *TauReal*
  ftupla
ftipus

D'aquesta manera la dimensió del vector es decideix en temps d'execució i s'emmagatzema en el camp `dim`. Només hi ha una restricció: la dimensió no pot superar el valor de la constant `MaxDim`. L'objectiu següent serà eliminar aquesta restricció. Per això a continuació es presenta una implementació de les operacions sobre el tipus `Vector` basada en l'ús de la memòria dinàmica.

Especificació

Per dissenyar algorismes que usin vectors suposarem que disposem del tipus `Vector` i de les operacions següents

tipus
  Vector
ftipus

{Prec: $n > 0$}
funcio *CrearVector*(**ent** n : **enter**) **retorna** *Vector*
{Post: *CrearVector*(n) és un vector amb interval [0..$n-1$]}

{Prec: **cert**}
funcio *DimVector*(**ent** v : *Vector*) **retorna enter**
{Post: *DimVector*(v) és la dimensió del vector v}

{Prec: $0 \leq i < DimVector(v)$}
accio *AssigVector*(**entsor** v : *Vector*, **ent** i : **enter**, **ent** r : **real**)
{Post: $v[i] = r$}

{Prec: $0 \leq i < DimVector(v)$}

funcio *ConsVector*(**ent** v : *Vector*, **ent** i : **enter**) **retorna real**
{Post: $ConsVector(v, i) = v[i]$}

{Prec: **cert**}
accio *DestruirVector*(**entsor** v : *Vector*)
{Post: v destruït}

L'acció `DestruirVector` és necessària per alliberar la memòria dinàmica emprada i evitar així d'exhaurir-la. El fitxer `vector.h`, llistat a l'apèndix A.32, conté la traducció a `C` d'aquesta especificació.

Implementació

El fitxer `vector.c`, llistat a l'apèndix A.31, conté la implementació en `C` de les operacions especificades a l'apartat anterior. Fixeu-vos especialment en la reserva d'espai en `CrearVector`, en l'alliberament d'espai en `DestruirVector` i en l'accés als components del vector en `AssigVector` i `ConsVector`. També cal destacar que es comprova mitjançcant assercions que l'accés al vector es fa dins de l'interval en què s'ha definit.

Exercici 51 *Dissenyeu un subprograma que, donats dos vectors en calculi el producte escalar usant les operacions sobre el tipus* `Vector` *presentades prèviament. Useu aquest subprograma en un algorisme que rebi pel canal estàndard d'entrada els dos vectors amb el format*

$$n\, x_0\, x_1 \ldots x_{n-1}$$

on n —enter— és el nombre d'elements del vector i x_i —real— són els components del vector. Els components són separats per un o més separadors —blancs, tabuladors, salts de línia. Pel canal estàndard de sortida ha d'escriure un real, el producte escalar dels dos vectors de l'entrada

Traduiu a `C` *l'algorisme, compileu i munteu el programa i executeu-lo per a diferents jocs de prova.*

Exercici 52 *Dissenyeu un subprograma que donats dos vectors en calculi el vector diferència usant les operacions sobre el tipus* `Vector` *presentades prèviament. Useu aquest subprograma en un algorisme similar al de l'exercici 51 que rebi dos vectors pel canal estàndard d'entrada i escrigui pel canal estàndard de sortida el vector diferència.*

Traduiu a `C` *l'algorisme, compileu i munteu el programa i executeu-lo per a diferents jocs de prova.*

Exercici 53 *Dissenyeu un nou joc d'operacions sobre el tipus* `Vector` *de manera que l'interval de variació de l'índex es doni en el moment de la seva creació. Per exemple,*

```
v = CrearVector(3, 10);
```

10.4.4 Exemple: matrius

El tipus `Matriu` és usat també àmpliament en aplicacions de càlcul numèric. El raonament que ens porta a considerar una representació del tipus `Matriu` basada en l'ús de memòria dinàmica és anàleg al del tipus `Vector` descrit en l'apartat 10.4.3. Per tant, passarem directament a especificar les operacions sobre aquest tipus.

Especificació

Suposarem l'existència del tipus `Matriu`. Les seves operacions es poden especificar com segueix:

tipus
  Matriu
ftipus

{Prec: $nf > 0 \ \wedge \ nc > 0$}
funcio *CrearMatriu*(**ent** nf, nc : **enter**) **retorna** *Matriu*
{Post: *CrearMatriu*(nf, nc) és una matriu amb intervals $[0..nf-1, 0..nc-1]$}

{Prec: *cert*}
funcio *FilesMatriu*(**ent** m : *Matriu*) **retorna enter**
{Post: *FilesMatriu*(m) és el nombre de files de la matriu m}

{Prec: *cert*}
funcio *ColsMatriu*(**ent** m : *Matriu*) **retorna enter**
{Post: *ColsMatriu*(m) és el nombre de columnes de la matriu m}

{Prec: $0 \leq i <$ *FilesMatriu*$(m) \ \wedge \ 0 \leq j <$ *ColsMatriu*(m)}
accio *AssigMatriu*(**entsor** m : *Matriu*, **ent** i, j : **enter**, **ent** r : **real**)
{Post: $m[i, j] = r$}

{Prec: $0 \leq i <$ *FilesMatriu*$(m) \ \wedge \ 0 \leq j <$ *ColsMatriu*(m)}
funcio *ConsMatriu*(**ent** m : *Matriu*, **ent** i, j : **enter**) **retorna real**
{Post: *ConsMatriu*$(m, i, j) = m[i, j]$}

{Prec: *cert*}
accio *DestruirMatriu*(**entsor** m : *Matriu*)
{Post: m destruïda}

El fitxer `matriu.h`, llistat a l'apèndix A.17, conté la traducció a `C` de la part d'especificació.

Implementació

El fitxer `matriu.c`, llistat a l'apèndix A.16, conté la implementació en `C` de les operacions especificades a l'apartat anterior. Observeu que en la funció `CrearMatriu` es reserva espai per a una taula amb tants elements com el producte del nombre de files pel de columnes. Estem usant una representació lineal d'una matriu sobre una taula. Això es té en compte quan s'accedeix a un element en les operacions `AssigMatriu` i `ConsMatriu`. La posició que ocupa dins la taula l'element a_{ij} és $i * n_c + j$ essent n_c el nombre de columnes. Igual que en el cas del tipus `Vector`, abans d'accedir a l'element a_{ij} es comprova que i i j estiguin dins l'interval de files i columnes, respectivament.

Exercici 54 *Dissenyeu un subprograma que calculi el producte de dues matrius usant les operacions sobre el tipus* `Matriu` *especificades prèviament. Useu aquest subprograma en un algorisme que rebi pel canal estàndard d'entrada dues matrius amb el format següent:*

$$f\ c\ a_{00}\ a_{01} \ldots a_{0c-1}\ a_{10}\ a_{11} \ldots a_{1c-1} \ldots a_{f-10}\ a_{f-11} \ldots a_{f-1c-1}$$

on f és el nombre de files, c és el nombre de columnes, tots dos enters, i els a_{ij} són els elements de la matriu, reals. Els element de la matriu són separats per un o més separadors —blanc, tabulador, salt de línia. L'algorisme haurà d'escriure pel canal estàndard de sortida la matriu producte amb el mateix format.

Traduïu a `C` *l'algorisme, compileu i munteu el programa i executeu-lo per a diferents jocs de prova.*

Exercici 55 *Dissenyeu un subprograma que, donats una matriu m i dos vectors v i w, calculi el vector $z = m \times v - w$ usant les operacions sobre els tipus* `Vector` *i* `Matriu` *presentades prèviament. Useu aquest subprograma en un algorisme similar al de l'exercici 54 que rebi m, v i w pel canal estàndard d'entrada i escrigui pel canal estàndard de sortida el vector z.*

Traduïu a `C` *l'algorisme, compileu i munteu el programa i executeu-lo per a diferents jocs de prova.*

Exercici 56 *Dissenyeu un nou joc d'operacions sobre el tipus* `Matriu` *de manera que l'interval de variació de les files i les columnes es doni en el moment de la seva creació. Per exemple,*

```
m = CrearMatriu(1, 5, 3, 10);
```

Exercici 57 *Inseriu a la vostra llibreria local (vegeu el capítol 9) les unitats de compilació separada* `vector.c` *i* `matriu.c`*. Compileu, munteu i executeu els programes dels exercicis 51 i 54 tenint en compte la nova organització. Recordeu que també cal moure els fitxers* `vector.h` *i* `matriu.h` *al vostre directori d'inclusions.*

Capítol 11

Complexitat d'un algorisme

11.1 Material necessari

- Disquet de treball.
- Calculadora.
- Paper i llapis.
- Fulls de paper mil·limetrat.

11.2 Objectius

Aquesta pràctica té com a principal objectiu descobrir quin és el comportament dels programes pel que fa al cost en temps d'execució. Com a objectius secundaris, també practicarem el procés d'edició, compilació i muntatge i treballarem amb esquemes fonamentals com són la cerca lineal i la cerca dicotòmica.

11.3 Introducció

Fonamentalment, es tracta de jugar amb dos programes que es troben escrits.El programa `lin1.c` és una aplicació que fa una cerca seqüencial en una taula d'enters. El programa `bin1.c` és una aplicació que fa una cerca binària en una taula d'enters.

Ambdós programes estan preparats per què puguin donar com a resultat el temps d'execució i també per què puguin simular màquines de diferenta velocitat.

En el punt següent, es descriuen els dos programes fent èmfasi en el seu aspecte algorítmic. Més endavant es comenten els aspectes relacionats amb el càlcul del temps. A continuació s'explica com compilar els programes de proves i, finalment, s'exposa un guió per a l'exercici que s'ha de fer.

L'exercici que es vol fer consta de dues parts. La primera cal fer-la a la sala de terminals. La segona cal dur-la a terme a casa, individualment o en grup.

11.4 Els programes

Els dos programes que es volen provar implementen un esquema de cerca sobre una taula d'enters. La taula d'enters es genera automàticament de manera ordenada i l'usuari pot decidir la mida (MAXIM=15000). Una vegada s'ha generat la taula, els programes cerquen un enter qualsevol que l'usuari dóna. Finalment els programes escriuen pel canal de sortida alguns missatges indicant si l'enter s'ha trobat i el temps que s'ha trigat a fer la cerca.

La diferència entre un i altre programa és l'algorisme emprat en fer la cerca. En el programa `lin1.c`, es fa servir un algorisme de cerca lineal. En el programa `bin1.c` s'ha emprat un algorisme de cerca binària. Els llistats dels dos programes es poden veure a l'apèndix.

Ambdós programes s'executen de la mateixa manera. Cal escriure'n el seu nom i subministrar pel canal estàndard d'entrada dos enters: el primer indica el nombre d'elements de la taula i el segon quin és l'enter que s'hi vol cercar. Així, si es vol fer una cerca lineal en una màquina amb 100 milisegons de temps d'instrucció, sobre una taula de 10000 elements i es vol que l'enter cercat sigui el 3562, cal escriure

```
    lin1

100 10000 3562
```

La resposta a aquest exemple serà un resum, escrit pel canal de sortida, on es diu si la cerca ha estat fructífera i el temps que ha trigat el càlcul.

11.5 El càlcul del temps

Per tal de dur a terme l'exercici, ens cal poder mesurar fidedignament el temps que el procés de cerca triga. Per fer-ho s'ha utilitzat una funció de la llibreria estàndard de ANSI-C anomenada `clock()`. Aquesta funció no te cap paràmetre i retorna, cada vegada que és cridada, un enter que indica quantes unitats de temps han transcorregut des de l'inici de l'execució. Les unitats de temps que aquesta funció utilitza són desconegudes, però se sap que si tenim N unitats de temps i les dividim per la constant `CLOCKS_PER_SEC` obtenim el temps mesurat en segons.

Cal adonar-se que el valor de la constant `CLOCKS_PER_SEC` també ens indica la mesura més fina que el rellotge pot fer. En el cas dels PC's, `CLOCKS_PER_SEC` val típicament de l'ordre de 100 i per tant la mesura més fina possible és d'uns 0.01 segons.

Per usar aquesta funció satisfactòriament en el moment de mesurar el temps que triga el càlcul `Q()`, cal fer el següent:

```
#include <time.h>
long temps;

temps = clock();
Q();
temps = clock() - temps;
```

és a dir, declarar una variable de tipus `long` (un enter de capacitat superior a la de `int`) on emmagatzemem el temps abans de cridar `Q()` —instant T_1. Després de la crida a `Q()` —instant T_2— calculem el temps emprat fent $T_2 - T_1$. Noteu que cal incloure els *headers* de `time.h` pel fet d'haver usat la funció `clock()`.

Després d'això, podem escriure el temps gastat en segons usant la constant de conversió, és a dir, fent una cosa similar a:

```
EscriureReal(((float)(temps)) / ((float)(CLOCKS_PER_SEC)));
```

Observeu la conversió d'enter a real afegida sobre la variable `temps` per tal d'obtenir un resultat real.

Fins aquí, tot sembla indicar que el temps que triga `Q()` és simple de calcular. Però hi ha un problema que cal tenir en compte: què passa quan el temps de `Q()` està per sota del llindar $\frac{1}{CLOCKS\_PER\_SEC}$? Doncs que no podem fer cap mesura! I, oh Llei de Murphy!, els temps d'execució dels programes de cerca que ens interessen són, sovint, inferiors a aquest llindar.

La solució a aquest problema no és senzilla ja que la finor del rellotge no es pot variar de cap manera. Per aquesta raó adoptem una solució diferent: fer que el cost en temps d'una instrucció sigui molt més gran del que és en realitat.

Per fer-ho, usarem el subprograma `delay(int milisegons)`. Aquesta acció que té com a paràmetre d'entrada un enter N. El seu funcionament consisteix a aturar l'execució durant N mil·lisegons. Amb això podem simular el cost en temps d'una sentència o conjunt de sentències. Imagineu que tenim les sentències `S1` i `S2` amb temps d'execució constant T_1 i T_2 milisegons. Aleshores podem construir la sentència següent:

```
S1;
S2;
delay(TEMPS_INSTRUCCIO);
```

on `TEMPS_INSTRUCCIO` és una expressió entera. Si us fixeu, el temps d'execució d'aquesta sentència serà de

$$\text{temps d'execució total} = \texttt{TEMPS\_INSTRUCCIO} + T_1 + T_2$$

Si ara teniu en compte que el temps d'execució d'una sentència (assignació, comparació, etc.) és, en general, d'alguns microsegons, els temps T_i resulten menyspreables enfront del valor `TEMPS_INSTRUCCIO` i, per tant, podem escriure

$$\begin{aligned} \text{temps d'execució total} &= \texttt{TEMPS\_INSTRUCCIO} + T_1 + T_2 \\ &\approx \texttt{TEMPS\_INSTRUCCIO} \end{aligned}$$

Amb això hem aconseguit substituir una seqüència de sentències per unes sentències equivalents que triguen molt més temps a executar-se. Temps, per altra banda, que nosaltres podem modificar canviant el valor de `TEMPS_INSTRUCCIO`. Per tant, amb aquesta modificació podem simular un computador que té un cost per instrucció qualsevol. Això ho aprofitarem per simular computadors més lents o més ràpids, segons les nostres necessitats.

Fins aquí s'ha dit tot el que fa referència a la manera de calcular el temps que triga una certa execució. Solament queda advertir que no totes les sentències poden ser alentides mitjançcant `delay()`. Com aquesta funció produeix un retard constant, solament sentències que tenen un temps constant d'execució poden ser alentides. En el cas que ens afecta, això és suficient.

11.6 Guió de la pràctica

1. Copieu els fitxers `lin1.c`, `bin1.c`, `delay.c`, `delay.h`, `entsort.c` i `entsort.h` i `Makefile` al vostre directori de treball.

2. Compileu i munteu els programes fent

   ```
   make -k
   ```

   El resultat d'aquest pas són dos programes executables anomenats `lin1` i `bin1`.

3. Ara cal prendre un conjunt de temps d'execució per a diferents proves. Essencialment cal obtenir el temps que triga una cerca usant cada un dels programes per a diverses mides de la taula de dades. Suposarem un computador amb un temps d'instrucció de 20 milisegons. Per disminuir els errors de mesura, farem més d'una prova per cada mida de la taula de dades.

   Així docs, i per a una taula donada de mida N, farem cinc cerques: les corresponents als enters 0, $\frac{N}{4}$, $\frac{N}{2}$, $\frac{3N}{4}$ i N. Repetirem el procés per a taules de mida N igual a 100, 500, 1000, 5000 i 10000. Amb totes aquestes mesures podrem omplir una taula com la 11.1.

   Una vegada sigui plena, cal omplir-ne una altra, però ara usant com a programa `bin1`.

4. Ja tenim les mesures de temps d'execució per als dos algorismes. Per continuar l'experiment, operarem simulant que comprem un computador el doble de ràpid. Per fer-ho, el valor del temps d'instrucció 10 milisegons.

   Tornarem a repetir les mesures del punt 2 amb el nou valor del temps d'instrucció (que simula una màquina molt més ràpida). Amb això haurem obtingut quatre taules plenes de

N =mida taula	Valor cercat				
	0	$\frac{N}{4}$	$\frac{N}{2}$	$\frac{3N}{4}$	N
100					
500					
1000					
5000					
10000					

Taula 11.1: Resultats de les cerques a `lin1`

N =mida taula	Valor cercat					Mitjana temps
	0	$\frac{N}{4}$	$\frac{N}{2}$	$\frac{3N}{4}$	N	
100						
500						
1000						
5000						
10000						

Taula 11.2: Mitja de les cerques a `lin1`

.

temps d'execució: dues per a cada programa d'exemple, una per a la màquina ràpida i una altra per a la màquina lenta.

5. Una vegada es tenen els temps d'execució cal, per a cada taula, fer la mitjana de les diverses cerques. És a dir, per a la taula corresponent al programa `lin1` executat sobre la màquina lenta, per exemple, cal calcular la mitjana dels temps dedicats a buscar els enters 0, 25, 50, 75 i 100 en la taula de 100 elements, els enters 0, 125, 250, 375 i 500 en la taula de 500 elements, etc. Els resultats d'aquestes mitjanes, els anotem al costat de la fila corresponent de la taula en qüestió. Amb això obtenim unes taules com la 11.2.

6. Quan tingueu totes les mitjanes calculades, construïu una taula com la 11.3. A cada casella, cal posar-hi la mitjana corresponent.

7. Sobre paper mil·limetrat, feu les quatre gràfiques corresponents a les quatre columnes amb quatre colors diferents. Si voleu, intenteu ajustar a cada corba un model de regressió. Comproveu que a les proves fetes amb el programa `lin1` s'ajusta bé un model lineal, mentre que a les fetes amb el programa `bin1` és el model logarítmic el que s'ajusta.

A partir de a les dades que heu obtingut, respongueu les preguntes següents tot intentant

N	Programa usat			
	`lin1`		`bin1`	
	Comp. ràpid	Comp. lent	Comp. ràpid	Comp. lent
100				
500				
1000				
5000				
10000				

Taula 11.3: Temps d'execució resumits

comprendre quines implicacions té l'experiment.

Exercici 58 *Si aconseguíssiu un algorisme de cerca lineal que fes una comparació menys que l'algorisme provat, quin tipus de millora en el temps d'execució obtindríeu?*

Exercici 59 *Suposeu que sou amos d'una empresa que disposa d'una aplicació tal que fa cerques en una base de dades en un temps proporcional a N, on N és el nombre d'elements de la taula on es cerca. Experimentalment, observeu que la competència té una aplicació semblant que fa la mateixa feina en un temps proporcional a $\log_2 N$. Com que la competència és millor i no veneu ni una escombra, heu de prendre una decisió. Quina decisió prendríeu: Compraríeu una màquina més ràpida? Invertiríeu temps millorant els vostres algorismes?*

Exercici 60 *Per quina raó sempre es diu que, en escriure un algorisme determinat, no és important el fet que aparegui "una assignació més o menys"?*

Exercici 61 *Si el fet que un algorisme contingui més o menys sentències no és massa important pel que fa al temps d'execució, quins són els factors que realment intervenen en el temps d'execució d'un algorisme?*

Exercici 62 *Intenteu deduir, de manera teòrica i a partir dels algorismes de cerca lineal i binària, el cost en temps que tenen en funció de la mida de la taula on es fa la cerca.*

Exercici 63 *Redacteu en un paper un comentari on, de manera ben estructurada, s'exposin els resultats de l'experiment, les conclusions a què heu arribat analitzant els resultats i com aquestes conclusions afecten en el moment de dissenyar un algorisme.*

Capítol 12

Tractament de fitxers de text

12.1 Material

- Disquet de treball
- Pautes de traducció a `ANSI-C`

12.2 Objectius

L'objectiu d'aquesta pràctica és la realització d'un programa que treballi amb més d'un fitxer seqüencial de text. Per fer-ho es proposa l'enunciat d'un problema i part de la seva resolució. Es demana que es finalitzi l'algorisme que soluciona el problema, s'implementi, es tradueixi i es provi.

12.3 Traducció de les operacions sobre fitxers

Per dur a terme aquesta pràctica és necessari conèixer com es tradueixen a `ANSI-C` les operacions sobre el tipus `FST` (fitxers seqüencials de text). De la mateixa manera que les operacions per llegir i escriure del canal d'entrada o sortida, cada operació sobre fitxers de text té una implementació ben definida en `C`. A la secció 12.7 trobareu totes les operacions definides a la classe de teoria implementades en `C`.

12.4 Descripció de l'aplicació

El programa que es vol fer treballarà amb dos fitxers de text. El primer fitxer, que anomenarem `carta.txt`, conté el text corresponent a una carta que s'ha d'enviar a un conjunt de persones. El segon fitxer, que anomenarem `llista.txt`, conté les dades corresponents a les persones a les qui els hem d'enviar les cartes. Direm que la informació corresponent a una persona és un registre. La informació d'un registre es compon de diferents camps. Per exemple el camp 1, farà referència al nom de cada persona.

El fitxer corresponent al text de la carta acaba amb el símbol `$`, el qual només serveix per designar final de fitxer, però no s'haurà d'imprimir. A més, en certs llocs de la carta apareixen símbols del tipus `#n`, on `n` és un número més gran que zero i menor o igual que el nombre de camps que té un registre. El símbol, `#n,` serveix per fer referència al camp n-èsim dels registres del segon fitxer.

El fitxer que conté les dades dels individus comenÇca amb un enter, que indica el nombre de camps que té cada registre. El nombre de camps ha de ser el mateix per a cada registre. Cada camp està separat del següent per un caràcter separador (espai en blanc, salt de línia o tabulador). El final de fitxer està determinat pel caràcter `$`.

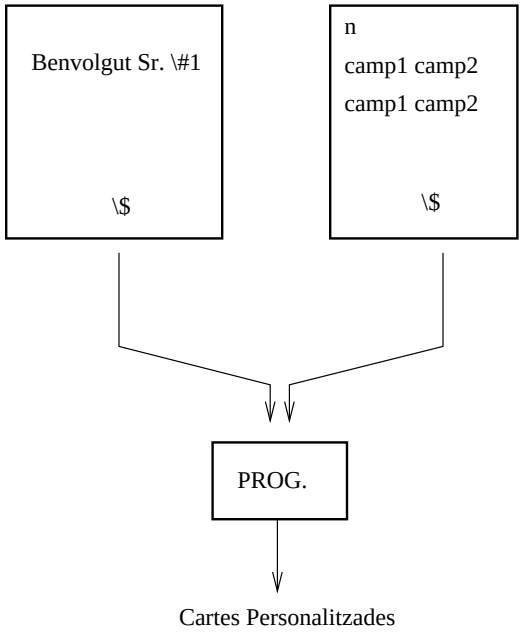

Figura 12.1: Esquema del programa

Es demana fer un programa tal que, donat un fitxer del tipus `carta.txt` i un fitxer del tipus `llista.txt`, imprimeixe una carta seguint el model del tipus donat, i personalitzada per cada un dels registres de `llista.txt` (vegeu 12.1).

12.5 Guió de la pràctica

El problema es dóna parcialment resolt. Dels primers nivells d'anàlisi descendent se'n dóna l'algorisme i dels nivells que manquen, la traducció. D'aquesta manera, l'única feina que queda és la traducció dels primers nivells d'anàlisi i la compilació de tot plegat.

1. Traduïu l'algorisme que es dóna a l'apartat 12.6 a C.
2. Copieu del disc del llibre el fitxer que conté la part traduïda. El nom de fitxer que conté aquests subprogrames és `mailing.c`.
3. Afegiu la part que vosaltres heu traduït al fitxer que heu copiat.
4. Compileu i munteu el programa que heu fet. Resoleu els errors que apareixen.
5. Creeu un fitxer de text del tipus `carta.txt` i un altre del tipus `llista.txt`. Poseu en aquest un mínim de quatre registres. Els noms dels fitxers han de ser els adequats per ser cridats posteriorment (vegeu amb quin nom s'obren a l'algorisme). Recordeu com s'ha definit el final de fitxer en cada cas.
6. Executeu el programa redireccionant la sortida cap a un fitxer i envieu a imprimir aquest fitxer.

12.6 Algorisme proposat

En aquesta secció trobaràs en part de l'algorisme per resoldre el problema proposat i també la definició de tipus utilitzada. En la secció següent trobareu la traducció dels subprogrames que,

juntament amb la traducció dels proposats aquí, donen una solució del problema.

Els algorismes i la definició de tipus proposats són

```
algoritme CartesPersonalitzades
  var
    individu : persona
    llista : FST
    n : enter
  fvar
  {Aplicarem l'esquema de recorregut al fitxer llista.txt}
  llista := ObrirFST(lectura,"llista.txt")
  LlegirEnter(llista, n)
  LlegirIndividu(llista, individu, n)
  mentre ¬DarrerIndividu(individu) fer
            GenerarCarta(individu)
            LlegirIndividu(llista, individu, n)
  fmentre
  TancarFST(llista)
falgoritme
```

Passem tot seguit a definir els tipus utilitzats en l'algorisme principal. La definició de les constants utilitzades es deixen per a l'estudiant.

```
tipus
  tau_car = taula [1..MAXLON] de caracter
  paraula = tupla
              mot : tau_car
              nc : enter
            ftupla
  tau_camp = taula [1..MAXC] de paraula
  persona = tupla
              camps : tau_camp
              ncamp : enter
            ftupla
ftipus
```

Tot seguit es donarà la implementació de les accions i funcions corresponents al nivell següent:

```
accio LlegirIndividu(entsor Dades : FST,
  sor individu : persona,
  ent n : enter)
  var
    i : enter
  fvar
  LlegirParaulaFST(Dades, individu.camps[1])
  si
     DarreraParaula(individu.camps[1]) → individu.ncamp := 0
  □ ¬(DarreraParaula(individu.camps[1])) →
       per i en [2..n] fer
            LlegirParaulaFST(Dades, individu.camps[i])
       fper
       individu.ncamp := n
  fsi
faccio
```

```
accio GenerarCarta(ent individu : persona)
  var
    carta : FST
    c : caracter
  fvar
  carta := ObrirFST(lectura,'carta.txt")
  LlegirCaracterFST(carta, c)
  mentre ¬(FinalFitxer(c)) fer
          TractarCaracter(c, carta, individu)
          LlegirCaracter(carta, c)
  fmentre
  TancarFST(carta)
  EscriureCaracter('Saltdepagina')
faccio


funcio DarrerIndividu(ent individu : persona) retorna boolea
  retorna (individu.ncamp = 0)
ffuncio
```

12.7 Pautes de traducció de les operacions sobre FST

En aquest apartat es donen les pautes de traducció a `ANSI-C` de les operacions sobre fitxers seqüencials de text emprades en aquest problema. Seguidament es comenten alguns aspectes que s'han considerat en escriure la traducció. La traducció literal la podreu trobar en els fitxers `fst.c`.

12.7.1 Aspectes d'implementació

Els següents aspectes de la traducció són interessants:

- Només s'usa la llibreria d'entrada/sortida estàndard de `ANSI-C`.
- El tipus FST és un apuntador. Quant a pas de paràmetres cal aplicar les mateixes pautes de traducció que per als tipus taula.
- La funció `ferror` s'avalua a cert si i solament si s'ha produït un error en la darrera operació d'entrada/sortida.
- La funció `fscanf` retorna el nombre de *tokens* que ha pogut llegir. Aquest valor de retorn serveix per detectar formats de dades incorrectes quan es llegeixen enters o reals.
- La funció `fprintf` retorna el nombre de caràcters escrits i un valor negatiu si s'ha produit algun error.
- Els `assert` es poden interpretar com la acció nul·la si el paràmetre s'avalua a **cert**. En cas contrari, avorta l'execució del programa.

Apèndix A

Llistats dels exemples

Aquest apèndix conté els llistats de tots els fitxers usats en el llibre. Aquests mateixos fitxers es poden obtenir del disc que acompanya el llibre, on es troben classificats per capítols.

A.1 Fitxer `bin1.c`

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

#include "delay.h"
#include "entsort.h"

typedef int t[30001];

int
main(void)
{
    int i;
    int nument;
    int enttriat;
    clock_t temps;
    int m, e, d;
    t tent;
    int TEMPS_INSTRUCCIO;

    LlegirEnter(&TEMPS_INSTRUCCIO);
    /* omplim la taula amb enters qualssevol */
    LlegirEnter(&nument);
    for (i = 0; i < nument; i++)
        tent[i] = i;
    /* afegim el sentinella */
    tent[i] = -1;

    /* fem la cerca de l'enter triat */
    LlegirEnter(&enttriat);

    temps = clock();

    e = 0;
    d = nument;
    while (e < d) {
        m = (e + d) / 2;
        if (tent[m] >= enttriat)
            d = m;
        else
            e = m + 1;
        delay(TEMPS_INSTRUCCIO);
    }
```

```
    temps = clock() - temps;

    /* escrivim els resultats */
    EscriureMissatge("\n\nRESULTATS DE LA CERCA:\n");
    if (tent[e] == enttriat) {
        EscriureMissatge("TROBAT: tent[");
        EscriureEnter(e);
        EscriureMissatge("]=");
        EscriureEnter(enttriat);
        EscriureCaracter('\n');
    } else {
        EscriureMissatge("NO TROBAT\n");
    }
    EscriureMissatge("Temps de cerca = ");
    EscriureReal(((float)(temps)) / ((float)(CLOCKS_PER_SEC)));
    EscriureMissatge(" segons\n");

    return EXIT_SUCCESS;

}
```

A.2 Fitxer `delay.c`

```
#include <time.h>

#include "delay.h"

void
delay(int t)
{
    clock_t a, b;

    a = clock();
    b = clock();
    while (((b - a) * 1000) / CLOCKS_PER_SEC < t) {
        b = clock();
    }
}
```

A.3 Fitxer `delay.h`

```
#ifndef DELAY_H
#define DELAY_H

void delay(int t);

#endif
```

A.4 Fitxer `entsort.c`

```
/*
 * entsort.c
 *      Operacions d'entrada/sortida
 */

/*
 * Inclusions estandard
```

```
 */
#include <assert.h>
#include <stdio.h>

/*
 * Inclusions
 */
#include "entsort.h"

/*
 * Implementacio dels subprogrames
 */
void
LlegirCaracter(char *const c)
{
    *c = getchar();
}

void
LlegirEnter(int *const i)
{
    int ret;

    ret = scanf("%d", i);
    assert(ret == 1);
}

void
LlegirReal(float *const f)
{
    int ret;

    ret = scanf("%f", f);
    assert(ret == 1);
}

void
EscriureCaracter(char c)
{
    putchar(c);
}

void
EscriureEnter(int i)
{
    int ret;

    ret = printf("%d", i);
    assert(ret > 0);
}

void
EscriureReal(float f)
{
    int ret;

    ret = printf("%g", f);
    assert(ret > 0);
}

void
EscriureMissatge(char c[])
{
    printf(c);
}
```

A.5 Fitxer `entsort.h`

```
/*
 * entsort.h
 *      Operacions d'entrada/sortida
 */

#ifndef ENTSORT_H
#define ENTSORT_H

void LlegirCaracter(char *const c);
void LlegirEnter(int *const i);
void LlegirReal(float *const f);

void EscriureCaracter(char c);
void EscriureEnter(int i);
void EscriureReal(float f);

void EscriureMissatge(char c[]);

#endif
```

A.6 Fitxer `fc.c`

```
/*
 * fc.c
 *      FraccioCanonica
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdlib.h>

/*
 * Inclusions
 */
#include "fc.h"

#include "natural.h"

/*
 * Implementacio dels subprogrames
 */
Racional
FraccioCanonica(Racional r)
{
    int nr, dr, mcdr;

    nr = NumeradorRacional(r);
    dr = DenominadorRacional(r);
    mcdr = mcd(abs(nr), abs(dr));
    return CrearRacional(nr / mcdr, dr / mcdr);
}
```

A.7 Fitxer `fc.h`

```
/*
 * fc.h
```

```
 *      FraccioCanonica
 */

#ifndef FC_H
#define FC_H

#include "racional.h"

Racional FraccioCanonica(Racional r);

#endif
```

A.8 Fitxer fst.c

```
/*
 * Implementacio de les operacions sobre FST
 */

#include <assert.h>
#include <stdio.h>
#include "fst.h"

/* Implementacio de les operacions */

FST
ObrirFST(ModeFST m, const char *nom)
{
    FST f;

    if (m == lectura) {
        f = (FST) fopen(nom, "r");
    } else if (m == escriptura) {
        f = (FST) fopen(nom, "w");
    } else
        assert(0);
    assert(f != NULL);
    return f;
}

void
LlegirEnterFST(FST f, int *const e)
{
    int ret;

    ret = fscanf((FILE *) f, "%d", e);
    assert(feof((FILE *) f) || (ret == 1 && !ferror((FILE *) f)));
}

void
LlegirRealFST(FST f, float *const r)
{
    int ret;

    ret = fscanf((FILE *) f, "%f", r);
    assert(feof((FILE *) f) || (ret == 1 && !ferror((FILE *) f)));
}

void
LlegirCaracterFST(FST f, char *const c)
{
    *c = fgetc((FILE *) f);
    assert(feof((FILE *) f) || !ferror((FILE *) f));
}

void
```

```
EscriureEnterFST(FST f, int e)
{
    int ret;

    ret = fprintf((FILE *) f, "%d", e);
    assert(ret >= 0 && !ferror((FILE *) f));
}

void
EscriureRealFST(FST f, float r)
{
    int ret;

    ret = fprintf((FILE *) f, "%g", r);
    assert(ret >= 0 && !ferror((FILE *) f));
}

void
EscriureCaracterFST(FST f, char c)
{
    char ret;

    ret = fputc(c, (FILE *) f);
    assert(ret == c && !ferror((FILE *) f));
}

int
FdFST(FST f)
{
    return feof((FILE *) f);
}

void
TancarFST(FST f)
{
    int ret;

    ret = fclose((FILE *) f);
    assert(ret == 0);
}
```

A.9 Fitxer `fst.h`

```
/*
 * Implementacio de les operacions sobre FST
 */

#ifndef FST_H
#define FST_H

/* Inclusions: llibreries estandard */

#include <stdio.h>
    typedef enum
{ lectura, escriptura
}
ModeFST;
typedef FILE *FST;
FST ObrirFST(ModeFST m, const char *nom);
void LlegirEnterFST(FST f, int *const e);
void LlegirRealFST(FST f, float *const r);
void LlegirCaracterFST(FST f, char *const c);
void EscriureEnterFST(FST f, int e);
void EscriureRealFST(FST f, float r);
void EscriureCaracterFST(FST f, char c);
```

```
int FdFST(FST f);
void TancarFST(FST f);


#endif  /*  */
```

A.10 Fitxer graus.c

```
/*************************************************************************/
/* Fitxer : graus.c                                                      */
/* Transforma graus, minuts i segons a radians.                          */
/* Donats tres reals pel canal estandart d'entrada que representen una   */
/* quantitat en graus, minuts i segons, calcula quants radians son.      */
/*************************************************************************/

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

#define PI 3.1415926            /* declaracio d'una constant */

/* programa principal */

float LlegirReal(void);
void EscriureReal(float);

int
main(void)
{
    /* declaracio de variables */
    float g, m, s, c, factor;

    /* inici codi del programa */
    g = LlegirReal();
    m = LlegirReal();
    s = LlegirReal();           /* lectura de valors */

    factor = PI / 180.0;
    c = g * factor + m * factor / 60.0 + s * factor / 3600.0;
    EscriureReal(c);

    return EXIT_SUCCESS;
}

float
LlegirReal(void)
{
    int ret;
    float r;

    ret = scanf("%f", &r);
    assert(ret == 1);
    return r;
}

void
EscriureReal(float r)
{
    printf("%g", r);
}

/* Fi del programa */
```

A.11 Fitxer graus1.c

```
/*************************************************************************/
/* Fitxer : graus1.c      AMB WARNINGS                                   */
/* Transforma graus, minuts i segons a radians.                          */
/* Donats tres reals pel canal estandart d'entrada que representen una   */
/* quantitat en graus, minuts i segons, calcula quants radians son.      */
/*************************************************************************/

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

#define PI 3.1415926            /* declaracio d'una constant */

/* programa principal */

float LlegirReal(void);
void EscriureReal(float);

int
main(void)
{
    /* declaracio de variables */
    float g, m, s, c, factor;
    float k;

    /* inici codi del programa */
    g = LlegirReal();
    m = LlegirReal();
    s = LlegirReal();           /* lectura de valors */

    factor == PI / 180.0;
    c = g * factor + m * factor / 60.0 + s * factor / 3600.0;
    EscriureReal(c);

    return EXIT_SUCCESS;
}

float
LlegirReal(void)
{
    int ret, j;
    float r;

    ret = scanf("%f", &r);
    assert(ret == 1);
    return r;
}

void
EscriureReal(char r)
{
    printf("%g", r);
}

/* Fi del programa */
```

A.12 Fitxer graus2.c

```
/*************************************************************************/
/* Fitxer : graus2.c          AMB ERRORS                                 */
/* Transforma graus, minuts i segons a radians.                          */
```

```
/* Donats tres reals pel canal estandart d'entrada que representen una   */
/* quantitat en graus, minuts i segons, calcula quants radians son.      */
/************************************************************************/

#include\<stdlib.h\>
#include <stdio.h>
#include <assert.h>

#define PI 3.1415926            /* declaracio d'una constant */

/* programa principal */

float
LlegirReal(void)
     void EscriureReal();

     int main(void)
{
    /* declaracio de variables */
    float g, m, s, c, factor;

    /* inici codi del programa */
    g = LlegirReal();
    m = LlegirReal();
    s = LlegirReal();           /* lectura de valors */

    factor = PI / 180.0
        c = g * factor + m * factor / 60.0 + s * factor / 3600.0;
    EscriureReal(c);

    return EXIT_SUCCESS;
}

float
LlegirReal(void)
{
    int ret;
    float r;

    ret = scanf("%f", &r);
    assert(ret == 1);
    return r;
}

void
EscriureReal(float r)
{
    printf("%g", r);
}

/* Fi del programa */
```

A.13 Fitxer graus3.c

```
/************************************************************************/
/* Fitxer : graus.c   AMB ERRORS DE MUNTATGE                            */
/* Transforma graus, minuts i segons, a radians.                        */
/* Donats tres reals pel canal estandart d'entrada que representen una   */
/* quantitat en graus, minuts i segons, calcula quants radians son.      */
/************************************************************************/

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

```
#define PI 3.1415926            /* declaracio d'una constant */

/* programa principal */

float LlegirReal(void);
void EscriureReal(float);

int
main(void)
{
    /* declaracio de variables */
    float g, m, s, c, factor;

    /* inici codi del programa */
    g = LlegirReal();
    m = LlegirReal();
    s = LlegirReal();           /* lectura de valors */

    factor = PI / 180.0;
    c = g * factor + m * factor / 60.0 + s * factor / 3600.0;
    EscriureReal(c);

    return EXIT_SUCCESS;
}

float
LlegirReal(void)
{
    int ret;
    float r;

    ret = scanf("%f", &r);
    assert(ret == 1);
    return r;
}

/* Fi del programa */
```

A.14 Fitxer `lin1.c`

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

#include "delay.h"
#include "entsort.h"

typedef int t[30001];

int
main(void)
{
    int i;
    int nument;
    int enttriat;
    clock_t temps;
    t tent;
    int TEMPS_INSTRUCCIO;

    LlegirEnter(&TEMPS_INSTRUCCIO);
    /* omplim la taula amb enters qualssevol */
    LlegirEnter(&nument);
    for (i = 0; i < nument; i++)
        tent[i] = i;
    /* afegim el sentinella */
    tent[i] = -1;
```

```
    /* fem la cerca de l'enter triat */
    LlegirEnter(&enttriat);

    temps = clock();

    i = 0;
    while (tent[i] != -1 && tent[i] != enttriat) {
        i = i + 1;
        delay(TEMPS_INSTRUCCIO);
    }

    temps = clock() - temps;

    /* escrivim els resultats */
    EscriureMissatge("\n\nRESULTATS DE LA CERCA:\n");
    if (tent[i] == enttriat) {
        EscriureMissatge("TROBAT: tent[");
        EscriureEnter(i);
        EscriureMissatge("]=");
        EscriureEnter(enttriat);
        EscriureCaracter('\n');
    } else {
        EscriureMissatge("NO TROBAT\n");
    }
    EscriureMissatge("Temps de cerca = ");
    EscriureReal(((float)(temps)) / ((float)(CLOCKS_PER_SEC)));
    EscriureMissatge(" segons\n");

    return EXIT_SUCCESS;

}
```

A.15 Fitxer `mailing.c`

```
/*
 * Part ja traduida de l'exemple del capitol de fitxers
 *
 */

void
LlegirParaulaFST(FST f, paraula * const nom)
{
    char c;

    LlegirCaracterFST(f, &c);
    SaltarSeparadors(f, &c);
    (*nom).nc = 0;

    while (!separador(c) && (c != '$')) {
        (*nom).nc = (*nom).nc + 1;
        (*nom).mot[(*nom).nc] = c;
        LlegirCaracterFST(f, &c);
    }
    if (c == '$') {
        (*nom).mot[1] = c;
    }
}

bool
DarreraParaula(paraula nom)
{
    return (nom.mot[1] == '$');
}
```

```
void
LlegirCaracterFST(FST f, char *const c)
{
    *c = fgetc((FILE *) f);
    assert(feof((FILE *) f) || !ferror((FILE *) f));
}

void
TractarCaracter(char *const c, FST f, persona ind)
{
    int n;

    if ((*c) != '#') {
        EscriureCaracter(*c);
    } else {
        LlegirEnterFST(f, &n);
        if (n > ind.ncamp || n <= 0) {
            EscriureMissatgeError();
        } else {
            EscriureCamp(ind.camps[n]);
        }
    }

}

void
SaltarSeparadors(FST f, char *const c)
{
    while (separador(*c)) {
        LlegirCaracterFST(f, c);
    }
}

bool
separador(char c)
{
    return ((c == ' ') || (c == '\t') || (c == '\n'));
}

void
EscriureCaracter(char c)
{
    putchar(c);
}

void
EscriureCamp(paraula nom)
{
    int i;

    for (i = 1; i <= nom.nc; i++) {
        EscriureCaracter(nom.mot[i]);
    }
}

bool
FinalFitxer(char c)
{
    return (c == '$');
}

void
EscriureMissatgeError(void)
{
    printf("T'has passat");
}
```

A.16 Fitxer matriu.c

```
/*
 * matriu.c
 *      Matriu de reals.
 */

#include <assert.h>
#include <stdlib.h>

#include "matriu.h"

/* Implementacio de les operacions */

Matriu
CrearMatriu(int nf, int nc)
{
    Matriu m;

    assert(nf > 0 && nc > 0);
    m.nf = nf;
    m.nc = nc;
    m.telem = (float *)malloc(nf * nc * sizeof (float));

    assert(m.telem != NULL);
    return m;
}

int
FilesMatriu(Matriu m)
{
    return m.nf;
}

int
ColsMatriu(Matriu m)
{
    return m.nc;
}

void
AssigMatriu(Matriu * const m, int i, int j, float f)
{
    assert(0 <= i && i < (*m).nf && 0 <= j && j < (*m).nc);
    (*m).telem[i * (*m).nc + j] = f;
}

float
ConsMatriu(Matriu m, int i, int j)
{
    assert(0 <= i && i < m.nf && 0 <= j && j < m.nc);
    return m.telem[i * m.nc + j];
}

void
DestruirMatriu(Matriu * const m)
{
    free((*m).telem);
}
```

A.17 Fitxer matriu.h

```
/*
 * matriu.h
 *      Matriu de reals.
 */

#ifndef MATRIU_H
#define MATRIU_H

/* Tipus Matriu */

typedef struct
{
    float *telem;               /* Taula d'elements */
    int nf;                     /* Nombre de files */
    int nc;                     /* Nombre de columnes */
}
Matriu;

/* Operacions */

Matriu CrearMatriu(int nf, int nc);
int FilesMatriu(Matriu m);
int ColsMatriu(Matriu m);
void AssigMatriu(Matriu * const m, int i, int j, float f);
float ConsMatriu(Matriu m, int i, int j);
void DestruirMatriu(Matriu * const m);

#endif
```

A.18 Fitxer `mdaccall.c`

```
/*
 * mdaccall.c
 *      Accedeix a memoria previament alliberada
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/*
 * Prototipus
 */
int *CrearEnter(void);
void DestruirEnter(int *i);

float *CrearReal(void);
void DestruirReal(float *f);

void EscriureReal(float f);

/*
 * Programa principal
 */
int
main(void)
{
    int *e;
    float *r;

    r = CrearReal();
```

```
    *r = 3.78;
    EscriureReal(*r);
    DestruirReal(r);
    e = CrearEnter();
    *e = 8999;
    EscriureReal(*r);
    DestruirEnter(e);

    return EXIT_SUCCESS;
}

/*
 * Implementacio dels subprogrames
 */
int *
CrearEnter(void)
{
    int *i;

    i = malloc(sizeof (int));

    assert(i != NULL);
    return i;
}

void
DestruirEnter(int *i)
{
    free(i);
}

float *
CrearReal(void)
{
    float *f;

    f = malloc(sizeof (float));

    assert(f != NULL);
    return f;
}

void
DestruirReal(float *f)
{
    free(f);
}

void
EscriureReal(float f)
{
    int ret;

    ret = printf("%g\n", f);
    assert(ret > 0);
}
```

A.19 Fitxer mdallnr.c

```
/*
 * mdallnr.c
 *      Allibera memoria no reservada previament.
 */
```

```
/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>

/*
 * Prototipus
 */
void EscriureEnter(int i);

char *CrearCaracter(void);
void DestruirCaracter(char *c);

/*
 * Programa principal
 */
int
main(void)
{
    int i, *ip;
    char *c;

    i = 3;
    ip = &i;
    EscriureEnter(i);
    free(ip);
    c = CrearCaracter();
    *c = 'a';
    DestruirCaracter(c);
    EscriureEnter(i);

    return EXIT_SUCCESS;
}

/*
 * Implementacio dels subprogrames
 */
void
EscriureEnter(int i)
{
    int ret;

    ret = printf("%d\n", i);
    assert(ret > 0);
}

char *
CrearCaracter(void)
{
    char *c;

    c = malloc(sizeof (char));

    assert(c != NULL);
    return c;
}

void
DestruirCaracter(char *c)
{
    free(c);
}
```

A.20 Fitxer mdelem.c

```
/*
 * mdelem.c
 *      Reserva i alliberament de memoria per tipus elementals.
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

/*
 * Inclusions
 */

/*
 * Prototipus
 */
char *CrearCaracter(void);
void DestruirCaracter(char *c);

int *CrearEnter(void);
void DestruirEnter(int *i);

float *CrearReal(void);
void DestruirReal(float *f);

bool *CrearBoolea(void);
void DestruirBoolea(bool * b);

void EscriureCaracter(char c);
void EscriureEnter(int i);
void EscriureReal(float f);

/*
 * Programa principal
 */
int
main(void)
{
    char *a;
    int *b;
    float *c;
    bool *d;

    /*
     * caracter
     */
    a = CrearCaracter();
    *a = 'x';
    EscriureCaracter(*a);
    EscriureCaracter('\n');
    DestruirCaracter(a);
    /*
     * enter
     */
    b = CrearEnter();
    *b = 123;
    EscriureEnter(*b + 3);
    EscriureCaracter('\n');
    DestruirEnter(b);
    /*
```

```
     * real
     */
    c = CrearReal();
    *c = 12.3;
    EscriureReal(*c - 0.3);
    EscriureCaracter('\n');
    DestruirReal(c);
    /*
     * boolea
     */
    d = CrearBoolea();
    *d = true;
    if (!*d && true) {
        EscriureCaracter('F');
    } else if (*d || false) {
        EscriureCaracter('C');
    } else
        assert(0);
    EscriureCaracter('\n');
    DestruirBoolea(d);

    return EXIT_SUCCESS;
}

/*
 * Implementacio dels subprogrames
 */
char *
CrearCaracter(void)
{
    char *c;

    c = malloc(sizeof (char));

    assert(c != NULL);
    return c;
}

void
DestruirCaracter(char *c)
{
    free(c);
}

int *
CrearEnter(void)
{
    int *i;

    i = malloc(sizeof (int));

    assert(i != NULL);
    return i;
}

void
DestruirEnter(int *i)
{
    free(i);
}

float *
CrearReal(void)
{
    float *f;

    f = malloc(sizeof (float));
```

```
    assert(f != NULL);
    return f;
}

void
DestruirReal(float *f)
{
    free(f);
}

bool *
CrearBoolea(void)
{
    bool *b;

    b = malloc(sizeof (bool));
    assert(b != NULL);
    return b;
}

void
DestruirBoolea(bool * b)
{
    free(b);
}

void
EscriureCaracter(char c)
{
    putchar(c);
}

void
EscriureEnter(int i)
{
    int ret;

    ret = printf("%d", i);
    assert(ret > 0);
}

void
EscriureReal(float f)
{
    int ret;

    ret = printf("%g", f);
    assert(ret > 0);
}
```

A.21 Fitxer mdnoall.c

```
/*
 * mdnoall.c
 *      Reserva memoria que no allibera mai.
 *      No es un algorisme. Acaba nomes quan es produeix un error en
 *      l'operacio de reserva de memoria.
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
```

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

/*
 * Definicio de tipus
 */

/*
 * Programa principal
 */
int
main(void)
{
    char *c;

    while (true) {
        c = malloc(1024 * sizeof (char));

        assert(c != NULL);
    }

    return EXIT_SUCCESS;
}
```

A.22 Fitxer `mdsino.c`

```
/*
 * mdsino.c
 *      Sinonims
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/*
 * Definicions de tipus
 */
typedef int TauEnter[10];

/*
 * Prototipus
 */
int *CrearEnter(void);
void DestruirEnter(int *i);

/*
 * Programa principal
 */
int
main(void)
{
    int i, *ip;
    TauEnter t;
    int *t3;
    int *p1, *p2;

    /*
     * i i *ip son sinonims
     */
    ip = &i;
    i = 3;
```

```
    printf(ï = %d, *ip = %d\n", i, *ip);
    *ip = 2;
    printf(ï = %d, *ip = %d\n", i, *ip);
    /*
     * t[3] i *t3 son sinonims
     */
    t[0] = 5;
    t[3] = 10;
    t3 = &t[3];
    printf("t[0] = %d, t[3] = %d, *t3 = %d\n", t[0], t[3], *t3);
    *t3 = 7;
    printf("t[0] = %d, t[3] = %d, *t3 = %d\n", t[0], t[3], *t3);
    /*
     * p1 i p2 apunten dos objectes diferents
     */
    p1 = CrearEnter();
    *p1 = 2;
    p2 = CrearEnter();
    *p2 = 3;
    printf("*p1(%p) = %d, *p2(%p) = %d\n", p1, *p1, p2, *p2);
    /*
     * p1 i p2 continuen apuntant dos objectes diferents pero amb
     * el mateix valor
     */
    *p1 = *p2;
    printf("*p1(%p) = %d, *p2(%p) = %d\n", p1, *p1, p2, *p2);
    /*
     * p1 i p2 son sinonims: apunten el mateix objecte.
     * L'objecte apuntat per p1 abans de l'assignacio
     * es una referencia penjada.
     */
    p1 = p2;
    printf("*p1(%p) = %d, *p2(%p) = %d\n", p1, *p1, p2, *p2);
    DestruirEnter(p1);
    /*
     * Error: alliberem 2 cops el mateix espai
     */
    DestruirEnter(p2);

    return EXIT_SUCCESS;
}

/*
 * Implementacio dels subprogrames
 */
int *
CrearEnter(void)
{
    int *i;

    i = malloc(sizeof (int));

    assert(i != NULL);
    return i;
}

void
DestruirEnter(int *i)
{
    free(i);
}
```

A.23 Fitxer mdtaula.c

```
/*
 * mdtaula.c
 *      Reserva i alliberament de memoria per tipus taula
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/*
 * Definicions de constants
 */
#define N 10
#define M (N + 10)

/*
 * Definicions de tipus
 */
typedef char TauCar[N];

typedef struct
{
    int nelem;
    TauCar telem;
}
Cadena;

typedef Cadena TauCadena[M];

/*
 * Prototipus
 */
TauCar *CrearTauCar(void);
void DestruirTauCar(TauCar * t);

TauCadena *CrearTauCadena(void);
void DestruirTauCadena(TauCadena * t);

void EscriureCaracter(char c);
void EscriureCadena(Cadena c);

/*
 * Programa principal
 */
int
main(void)
{
    TauCar *tc;
    TauCadena *tcad;

    /*
     * TauCar
     */
    tc = CrearTauCar();
    (*tc)[0] = 'a';
    (*tc)[5] = 'b';
    EscriureCaracter((*tc)[0]);
    EscriureCaracter((*tc)[5]);
    EscriureCaracter('\n');
    DestruirTauCar(tc);
    /*
     * TauCadena
     */
    tcad = CrearTauCadena();
```

```
    (*tcad)[0].nelem = 3;
    (*tcad)[0].telem[0] = 'x';
    (*tcad)[0].telem[1] = 'y';
    (*tcad)[0].telem[2] = 'z';
    (*tcad)[2].nelem = 4;
    (*tcad)[2].telem[0] = 'a';
    (*tcad)[2].telem[1] = 'b';
    (*tcad)[2].telem[2] = 'c';
    (*tcad)[2].telem[3] = 'd';
    EscriureCadena((*tcad)[0]);
    EscriureCadena((*tcad)[2]);
    DestruirTauCadena(tcad);

    return EXIT_SUCCESS;
}

/*
 * Implementacio dels subprogrames
 */
TauCar *
CrearTauCar(void)
{
    TauCar *t;

    t = malloc(sizeof (TauCar));
    assert(t != NULL);
    return t;
}

void
DestruirTauCar(TauCar * t)
{
    free(t);
}

TauCadena *
CrearTauCadena(void)
{
    TauCadena *t;

    t = malloc(sizeof (TauCadena));
    assert(t != NULL);
    return t;
}

void
DestruirTauCadena(TauCadena * t)
{
    free(t);
}

void
EscriureCaracter(char c)
{
    putchar(c);
}

void
EscriureCadena(Cadena c)
{
    int i;

    i = 0;
    while (i < c.nelem) {
        EscriureCaracter(c.telem[i]);
        i = i + 1;
    }
```

```
    EscriureCaracter('\n');
}
```

A.24 Fitxer `mdtupla.c`

```
/*
 * mdtupla.c
 *      Reserva i alliberament de memoria per tipus tupla
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/*
 * Definicio de constants
 */
#define M 10

/*
 * Definicio de tipus
 */
typedef int TaulaEnter[M];

typedef struct
{
    char c;
    int i;
    TaulaEnter t;
}
TuplaA;

typedef struct
{
    float x, y;
}
TuplaB;

/*
 * Prototipus
 */
TuplaA *CrearTuplaA(void);
void DestruirTuplaA(TuplaA * t);
void EscriureTuplaA(TuplaA t);

TuplaB *CrearTuplaB(void);
void DestruirTuplaB(TuplaB * t);
void EscriureTuplaB(TuplaB t);

void EscriureCaracter(char c);
void EscriureEnter(int i);
void EscriureReal(float f);

/*
 * Programa principal
 */
int
main(void)
{
    TuplaA *ta;
    TuplaB *tb;
    int i;
```

```
    /*
     * TuplaA
     */
    ta = CrearTuplaA();
    (*ta).c = 'v';
    (*ta).i = 321;
    i = 0;
    while (i < M) {
        (*ta).t[i] = 2 * i;
        i = i + 1;
    }
    EscriureTuplaA(*ta);
    DestruirTuplaA(ta);
    /*
     * TuplaB
     */
    tb = CrearTuplaB();
    (*tb).x = 3.7;
    (*tb).y = -0.1e-4;
    EscriureTuplaB(*tb);
    DestruirTuplaB(tb);

    return EXIT_SUCCESS;
}

/*
 * Implementacio dels subprogrames
 */
TuplaA *
CrearTuplaA(void)
{
    TuplaA *t;

    t = malloc(sizeof (TuplaA));
    assert(t != NULL);
    return t;
}

void
DestruirTuplaA(TuplaA * t)
{
    free(t);
}

void
EscriureTuplaA(TuplaA t)
{
    int i;

    EscriureCaracter(t.c);
    EscriureCaracter(' ');
    EscriureEnter(t.i);
    EscriureCaracter(' ');
    i = 0;
    while (i < M) {
        EscriureEnter(t.t[i]);
        EscriureCaracter(' ');
        i = i + 1;
    }
    EscriureCaracter('\n');
}

TuplaB *
CrearTuplaB(void)
{
    TuplaB *t;
```

```
    t = malloc(sizeof (TuplaB));
    assert(t != NULL);
    return t;
}

void
DestruirTuplaB(TuplaB * t)
{
    free(t);
}

void
EscriureTuplaB(TuplaB t)
{
    EscriureCaracter('(');
    EscriureReal(t.x);
    EscriureCaracter(',');
    EscriureReal(t.y);
    EscriureCaracter(')');
    EscriureCaracter('\n');
}

void
EscriureCaracter(char c)
{
    putchar(c);
}

void
EscriureEnter(int i)
{
    int ret;

    ret = printf("%d", i);
    assert(ret > 0);
}

void
EscriureReal(float f)
{
    int ret;

    ret = printf("%g", f);
    assert(ret > 0);
}
```

A.25 Fitxer `mitja.c`

```
/*************************************************************************/
/* Fitxer : MITJA.C                                                      */
/* Calcula la mitjana aritmetica d'un conjunt de nombres                 */
/* entrats pel teclat                                                    */
/*************************************************************************/

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

/* aixo es un comentari */

int LlegirEnter(void);
void EscriureReal(float);

int
```

```
main(void)
{                               /* rutina principal del programa, sempre hi ha
                                 * de ser */
    /* inici declaracio de variables */
    int n, sum, num;
    float rsum, rnum;

    /* inici codi del programa */
    n = LlegirEnter();

    sum = 0;
    num = 0;

    while (n != 0) {

        sum = sum + n;
        num = num + 1;
        n = LlegirEnter();
    }
    rsum = (float)sum;
    rnum = (float)num;          /* passem a reals els valors enters num i sum */

    if (num != 0) {
        EscriureReal(rsum / rnum);
        /*
         * traiem per pantalla el valor de la divisio entre rsum i
         * rnum, si rnum es diferent de zero
         */
    } else if (num == 0) {
        /* no fer res */
    } else
        assert(0);

    return EXIT_SUCCESS;
}

/* implementacio de les accions i funcions */

int
LlegirEnter(void)
{
    int e, ret;

    ret = scanf("%d", &e);
    assert(ret == 1);
    return e;
}

void
EscriureReal(float r)
{
    printf("%g", r);
}
```

A.26 Fitxer natural.c

```
/*
 * natural.c
 *      implementacio de les funcions sobre els nombres naturals
 */

/*
 * Inclusions estandard
 */
```

```
#include <assert.h>

/*
 * Inclusions
 */
#include "natural.h"

/*
 * Implementacio dels subprogrames
 */
int
mcd(int a, int b)
{
    int x, y;

    assert(a > 0 && b > 0);
    x = a;
    y = b;
    while (x != y) {
        if (x > y) {
            x = x - y;
        } else if (x < y) {
            y = y - x;
        } else
            assert(0);
    }
    return x;
}
```

A.27 Fitxer `natural.h`

```
/*
 * natural.h
 *      funcions sobre els nombres naturals
 */

#ifndef NATURAL_H
#define NATURAL_H

/*
 * Prototipus
 */
int mcd(int a, int b);

#endif
```

A.28 Fitxer `racional.c`

```
/*
 * racional.c
 *      implementacio de les operacions sobre els racionals
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdlib.h>
#include <stdbool.h>

/*
```

```
 * Inclusions
 */
#include "racional.h"
#include "natural.h"

/*
 * Implementacio dels subprogrames
 */
Racional
CrearRacional(int num, int den)
{
    Racional r;

    r.numerador = num;
    r.denominador = den;
    return r;
}

Racional
ZeroRacional(void)
{
    return CrearRacional(0, 1);
}

Racional
SumaRacional(Racional r, Racional s)
{
    int mcdrs;
    int smn, smd;

    mcdrs = mcd(abs(r.denominador), abs(s.denominador));
    smn = (r.numerador * s.denominador + s.numerador * r.denominador) / mcdrs;
    smd = r.denominador * s.denominador / mcdrs;
    return CrearRacional(smn, smd);
}

int
NumeradorRacional(Racional r)
{
    return r.numerador;
}

int
DenominadorRacional(Racional r)
{
    return r.denominador;
}

bool
IgualRacional(Racional r, Racional s)
{
    return r.numerador * s.denominador == s.numerador * r.denominador;
}
```

A.29 Fitxer racional.h

```
/*
 * racional.h
 *      operacions sobre els racionals
 */

#ifndef RACIONAL_H
#define RACIONAL_H
```

```
#include <stdbool.h>

/*
 * Inclusions
 */

/*
 * Definicio de tipus
 */
typedef struct
{
    int numerador, denominador;
}
Racional;

/*
 * Prototipus
 */
Racional CrearRacional(int num, int den);
Racional ZeroRacional(void);
Racional SumaRacional(Racional r, Racional s);
int NumeradorRacional(Racional r);
int DenominadorRacional(Racional r);
bool IgualRacional(Racional r, Racional s);

#endif
```

A.30 Fitxer `seqrac.c`

```
/*
 * seqrac.c
 *      Calcula la suma d'una sequencia de racionals acabada en 0/1
 */

/*
 * Inclusions estandard
 */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/*
 * Inclusions
 */
#include "racional.h"

/*
 * Prototipus
 */
static Racional LlegirRacional(void);
static void EscriureRacional(Racional r);

static int LlegirEnter(void);
static void EscriureEnter(int e);
static void EscriureCaracter(char c);

/*
 * Programa principal
 *      Prec: CSE = r1 r2 ... rn rf /\ rf = 0/1 /\ n >= 0
 *      Post: CSS = s /\ s = (Suma i: 1 <= i <= n: ri)
 */
int
main(void)
{
    Racional r;                 /* r = en_curs(CSE) */
```

```
    Racional suma;              /* suma = (Suma i: 1 <= i < pos(CSE, en_curs(CSE))) */

    /* Inicialitzacio del tractament */
    suma = ZeroRacional();
    /* Primer Element */
    r = LlegirRacional();
    /* Mentre no sentinella(en_curs(CSE)) */
    while (!IgualRacional(r, ZeroRacional())) {
        /* Tractament */
        suma = SumaRacional(suma, r);
        /* Seguent Element */
        r = LlegirRacional();
    }
    /* Finalitzacio del tractament */
    EscriureRacional(suma);

    return EXIT_SUCCESS;
}

/*
 * Implementacio dels subprogrames
 */
static Racional
LlegirRacional(void)
{
    int n, d;

    n = LlegirEnter();
    d = LlegirEnter();
    return CrearRacional(n, d);
}

static void
EscriureRacional(Racional r)
{
    EscriureEnter(NumeradorRacional(r));
    EscriureCaracter('/');
    EscriureEnter(DenominadorRacional(r));
    EscriureCaracter('\n');
}

static int
LlegirEnter(void)
{
    int ret, e;

    ret = scanf("%d", &e);
    assert(ret == 1);
    return e;
}

static void
EscriureEnter(int e)
{
    int ret;

    ret = printf("%d", e);
    assert(ret > 0);
}

static void
EscriureCaracter(char c)
{
    putchar(c);
}
```

A.31 Fitxer `vector.c`

```
/*
 * vector.c
 *      Vector de reals.
 */

#include <assert.h>
#include <stdlib.h>

#include "vector.h"

/* Implementacio de les operacions */

Vector
CrearVector(int n)
{
    Vector v;

    assert(n > 0);
    v.dim = n;
    v.telem = (float *)malloc(n * sizeof (float));

    assert(v.telem != NULL);
    return v;
}

int
DimVector(Vector v)
{
    return v.dim;
}

void
AssigVector(Vector * const v, int i, float f)
{
    assert(0 <= i && i < (*v).dim);
    (*v).telem[i] = f;
}

float
ConsVector(Vector v, int i)
{
    assert(0 <= i && i <= v.dim);
    return v.telem[i];
}

void
DestruirVector(Vector * const v)
{
    free((*v).telem);
}
```

A.32 Fitxer `vector.h`

```
/*
 * vector.h
 *      Vector de reals.
 */

#ifndef VECTOR_H
```

```
#define VECTOR_H

/* Tipus Vector */

typedef struct
{
    float *telem;               /* Taula d'elements */
    int dim;                    /* Dimensio */
}
Vector;

/* Operacions */

Vector CrearVector(int n);
int DimVector(Vector v);
void AssigVector(Vector * const v, int i, float f);
float ConsVector(Vector v, int i);
void DestruirVector(Vector * const v);

#endif
```

Apèndix B

Comandes de l'EMACS

COMANDA	INTERPRETACIÓ
`emacs fitxer`	Comença l'edició
`<Ctrl> X <Ctrl> C`	Acaba l'edició demanant conformitat per guardar el fitxer
`<Ctrl> X <Ctrl> S`	Guarda el fitxer que s'està editant
`<Ctrl> X <Ctrl> W`	Guarda el *buffer* en un fitxer especificat
`<Ctrl>G`	Aborta qualsevol comanda
<Ctrl>P	Puja el cursor una línia
<Ctrl>N	Baixa el cursor una línia
<Ctrl>B	Mou el cursor a l'esquerra
<Ctrl>F	Mou el cursor a la dreta
<Ctrl>E	Mou el cursor al final de línia
<Ctrl>A	Mou el cursor a l'inici de línia
<Ctrl>V	Mou el cursor a la pàgina següent
<Esc>V	Mou el cursor a la pàgina anterior
<Esc> <	Mou el cursor a l'inici del fitxer
<Esc> >	Mou el cursor al final del fitxer
`<Esc>x goto-line`	Va a una línia concreta
`<Ctrl> D`	Esborra el caràcter de sobre el cursor
`<Ctrl> W`	Esborra un text prèviament seleccionat
`<Ctrl> K`	Esborra el text des del cursor al final de línia
`<Ctrl> Y`	Còpia el text guardat en un *buffer*
`<Ctrl> <Espai>`	Posa una marca en la posició del cursor
`<Ctrl> _`	Anul·la la darrera comanda que s'ha executat
`<Ctrl>x u`	Anul·la el grup d'accions més recents
`<Ctrl> X <Ctrl> F`	Visita un fitxer i es visualitza
`<Ctrl> X <Ctrl> R`	Visita un fitxer en mode de lectura
`<Ctrl> X b`	Incorpora el *buffer* especificat a la finestra de treball
`<Ctrl> S`	Cerca incremental endavant d'una cadena de caràcters
`<Ctrl> R`	Cerca incremental endarrere d'una cadena de caràcters
`<Ctrl> G`	Interromp la comanda actual

Apèndix C

Resum de comandes del sistema operatiu

COMANDA	**INTERPRETACIÓ**
`cd`	Canvia el directori per defecte
`cp`	Copia el fitxer
`rm`	Esborra el fitxer
`rm -fr`	Esborra tot el directori
`ls`	Mostra els fitxers que hi ha en un directori
`emacs`	Executa l'editor
`grep`	Cerca un text en un fitxer
`mformat`	Formata un disc
`mkdir`	Crea un directori
`mv`	Mou un fitxer d'un directori a un altre
`lpr`	Imprimeix un fitxer
`rmdir`	Esborra un directori
`mv`	Canvia el nom d'un fitxer
`sort`	Ordena una seqüència de línies
`cat`	Mostra el contingut d'un fitxer per pantalla
`cp -R`	Copia jerarquies de subdirectoris

Apèndix D

Notació algorísmica i traducció a C

D.1 Introducció

A continuació es descriu la sintaxi de la notació algorísmica i es donen les pautes per a la seva traducció al llenguatge de programació `C`. La descripció sintàctica s'ha trencat en blocs que corresponen als apartats. Molts d'aquests apartats comencen amb un quadre on la columna de l'esquerra correspon a la sintaxi d'una construcció en notació algorísmica i la de la dreta, a la traducció a `C` de la mateixa construcció. Normalment hi ha una correspondència línia a línia entre els símbols de la columna esquerra i els de la dreta. Si no es dóna aquesta correspondència, en els subapartats que segueixen se n'explica el motiu. Segueixen aquest quadre una sèrie de subapartats amb comentaris al respecte.

La sintaxi es presenta usant una notació informal derivada de la notació *BNF*. Per la tipografia es poden distingir tres menes de símbols: els *símbols noterminals*, els terminals i les **paraules clau** en notació algorísmica o les `paraules clau` en `C`. A més, també apareixen altres símbols terminals com +, -, :, =, ¡ ... Un símbol noterminal T denota que en aquell punt pot aparèixer una construcció sintàctica que es descriurà més endavant en un quadre encapçalat per $T ::=$. Els símbols terminals corresponen a valors o a identificadors —noms de variables, de tipus, de subprogrames ... — i la seva sintaxi es descriu en apartats específics.

D.2 Algorisme

algorisme ::=	*algorisme ::=*
algorisme nom_algorisme	
	inclusions
definició_constants	*definició_constants*
definició_tipus	*definició_tipus*
	prototipus_accions_funcions
	`void main( void ) {`
declaració_variables	*declaració_variables*
sentències	*sentències*
falgorisme	`}`
accions_funcions	*accions_funcions*

D.2.1 àmbit de visibilitat

- Les definicions de constants i tipus són visibles en la declaració de variables, les sentències i en totes les accions i funcions de l'algorisme.

- Les variables declarades a la part de declaració de variables són visibles només en les sentències. No són visibles dins les accions i funcions.
- Les accions i funcions implementades junt amb l'algorisme són visibles dins les sentències i les accions i les funcions de l'algorisme.

D.2.2 Llenguatge C

- La part d'*inclusions* ha de contenir la inclusió dels fitxers que calgui de la llibreria estàndard
  - Sempre

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

  - Quan s'usi funcions d'entrada/sortida estàndard `getchar`, `scanf`, `printf` ... o el tipus `FILE` i les operacions sobre fitxers `fopen`, `fread`, `fwrite` ...

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

  - Quan s'usin funcions que treballen amb cadenes de caràcters: `strcpy`, `strcmp`, `strlen` ...

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

  - Quan s'usin funcions matemàtiques: `exp`, `log`, `sin`, `fabs` ...

    ```
    #include <math.h>
    ```

  - Quan s'usi el tipus booleà

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

D.3 Definició de constants

definició_constants ::=	*definició_constants ::=*
const nom_constant *:* nom_tipus = *expr_constant* ... **fconst**	`#define` nom_constant *expr_constant* ...

D.4 Definició de tipus

definició_tipus ::=	*definició_tipus ::=*
tipus *nou_tipus* ... **ftipus**	*nou_tipus* ...

nou_tipus ::=	*nou_tipus ::=*
nom_nou_tipus = *tipus_elemental* nom_nou_tipus = **taula** *[interval , interval ...]* **de** nom_tipus nom_nou_tipus = **tupla** nom_camp, nom_camp, ... *:* nom_tipus ... **ftupla** nom_nou_tipus = *(ident, ... , ident)*	`typedef` *tipus_elemental* nom_nou_tipus; `typedef` nom_tipus nom_nou_tipus `[` *num_elem* `]` `[` *num_elem* `]` ... ; `typedef struct {` nom_tipus nom_camp, nom_camp, ... ; ... `}` nom_nou_tipus; `typedef enum` `{` *ident*, ... , *ident* `}` nom_nou_tipus;

tipus_elemental ::=	*tipus_elemental ::=*
enter **real** **caràcter** **booleà**	`int` `float` `char` `bool`

interval ::=	
exp_constant_entera .. exp_constant_entera	

D.4.1 Llenguatge C

- Els intervals de les taules comencen en l'índex 0 i van fins a *num_elem* −1.
- Per definir una taula que pugui emmagatzemar el mateix nombre d'elements que en l'interval *primer_interval..darrer_interval* ha de ser *num_elem* = *darrer_interval* − *primer_interval* +1

D.5 Declaració de variables

declaració_variables ::= **var** nom_variable, nom_variable, . . . : nom_tipus . . . **fvar**	*declaració_variables ::=* nom_tipus nom_variable, nom_variable, . . . ; . . .

D.6 Sentències

sentència ::= *assignació* *condicional* *iterativa* *iterador_per* *crida_acció*	*sentència ::=* *assignació* *condicional* *iterativa* *iterador_per* *crida_acció*
assignació ::= nom_variable := *expressió*	*assignació ::=* nom_variable = *expressió*;
condicional ::= **si** *expressió_booleana1* ⟶ *sentències1* *expressió_booleana2* ⟶ *sentències2* . . . *fsi*	*condicional ::=* `if ( `*expressió_booleana1*` ) {` *sentències1* `} else if ( `*expressió_booleana2*` ) {` *sentències2* . . . `} else assert( 0 );`
iterativa ::= **mentre** *expressió_booleana* **fer** *sentències* **fmentre**	*iterativa ::=* `while ( `*expressió_booleana*` ) {` *sentències* `}`
iterador_per ::= **per** nom_var_entera **en** *[interval_variable]* **fer** *sentències* **fper**	*iterador_per ::=* `for (`nom_var_entera `=` *exp_primer_interval*; nom_var_entera `<=` *exp_darrer_interval*; nom_var_entera `=` nom_var_entera `+ 1) {` *sentències* `}`
iterador_per ::= **per** nom_var_entera **en** *[interval_variable]* **pas** *exp_entera* **fer** *sentències* **fper**	*iterador_per ::=* `for (`nom_var_entera `=` *exp_primer_interval* ; nom_var_entera `<=` *exp_darrer_interval*; nom_var_entera `=` nom_var_entera `+` *exp_entera*`) {` *sentències* `}`
interval_variable ::= *exp_primer_interval .. exp_darrer_interval*	
crida_acció ::= nom_acció*(paràmetre_actual,* *paràmetre_actual, . . .)*	*crida_acció ::=* nom_acció`(` *paràmetre_actual*`,` *paràmetre_actual*`, . . . );`
paràmetre_actual ::= nom_variable *expressió*	*paràmetre_actual ::=* nom_variable `&`nom_variable *expressió*

D.6.1 Notació algorísmica

- Assignació
  - És permesa qualsevol assignació sempre que hi hagi concordança de tipus entre l'expressió i la variable. Per tant, considerem correctes assignacions entre objectes de tipus taula o tupla.

D.6.2 Llenguatge C

- Assignació
  - Qualsevol assignació és permesa llevat que els tipus de l'expressió i de la variable siguin taules. En aquest cas, el compilador no generarà un error però l'assignació no funcionarà correctament —no produeix una còpia d'una regió de memòria sobre una altra sinó una còpia d'apuntadors.
- Condicional
  - La darrera línia de la sentència condicional, `} else assert( 0 );`, força que, com a mínim, una de les expressions booleanes sigui certa.
- Paràmetres actuals
  - El primer cas s'ha d'usar quan el paràmetre formal corresponent és d'entrada i la variable no és de tipus taula, és a dir, és de tipus elemental o de tipus tupla, o bé quan el paràmetre formal corresponent és d'entrada, sortida o entrada/sortida i la variables és de tipus taula.
  - El segon cas s'usa quan el paràmetre formal corresponent és de sortida o d'entrada/sortida i la variables no és de tipus taula, és a dir, és de tipus elemental o bé de tipus tupla.
  - El tercer cas s'usa quan el paràmetre formal corresponent és d'entrada i no és de tipus taula.
  - **ATENCIÓ!** La variable pot ser, en particular, de la forma `*nom_variable`. Cal recordar que els operadors `&` i `*` són l'invers l'un de l'altre. Per tant, si després d'aplicar les regles anteriors apareix `&*nom_variable`, l'hem de substituir per `nom_variable`.

D.7 Expressions

expressió ::=	*expressió ::=*
(expressió)	(*expressió*)
valor	valor
nom_variable	nom_variable
nom_constant	nom_constant
operador_unari expressió	*operador_unari expressió*
expressió operador_binari expressió	*expressió operador_binari expressió*
expressió[*expr_entera, expr_entera, ...*]	*expressió*[*expr_entera*][*expr_entera*] ...
expressió.nom_camp	*expressió*.nom_camp
nom_funció(*expressió, expressió, ...*)	nom_funció(*expressió*, *expressió*, ...)

operador_unari ::=	*operador_unari ::=*
–	-
no	*!*

operador_binari ::=	*operador_binari ::=*
$+$	`+`
$-$	`-`
$*$	`*`
$/$	`/`
div	`/`
mod	`%`
$<$	`<`
$>$	`>`
$=$	`==`
$\neq$	`!=`
$\leq$	`<=`
$\geq$	`>=`
i	`&&`
o	`||`

D.7.1 Notació algorísmica

- Funcions de conversió de tipus.

  {Prec: **cert**}
  funcio *truncar*(**ent** r : **real**) **retorna enter**
  {Post: $truncar(r) = \begin{cases} \lfloor r \rfloor & \text{si } r \geq 0 \\ \lceil r \rceil & \text{si } r \leq 0 \end{cases}$ }
  {Prec: **cert**}
  funcio *arrodonir*(**ent** r : **real**) **retorna enter**
  {Post: $arrodonir(r) = \lfloor r + 0.5 \rfloor$}
  {Prec: **cert**}
  funcio *EnterReal*(**ent** e : **enter**) **retorna real**
  {Post: $EnterReal(e) = e$}

- Les **prioritats dels operadors**, de major a menor, són

., [ ]	accés a tuples i taules
$-$, **no**	canvi de signe i negació lògica
$*$, $/$, **div**, **mod**	operadors multiplicatius
$+$, $-$	operadors additius
$<$, $>$, $\leq$, $\geq$, $=$, $\neq$	operadors relacionals
i	i lògica
o	o lògica

- L'**associativitat** entre operadors de la mateixa prioritat és d'esquerra a dreta.

D.7.2 Llenguatge C

- Les conversions de tipus tenen, en general, el format següent:

  ((tipus_destinació)(exp_a_convertir))

  La traducció de les operacions de conversió de tipus és

```
#include <math.h>

int truncar(float r)
{
    int e;

    if (r >= 0) {
```

```
        e = floor(r);
    } else {
        e = ceil(r);
    }
    return e;
}

int arrodonir(float r)
{
    return floor(r + 0.5);
}

int EnterReal(int e)
{
    return ((float)(e));
}
```

- Les **prioritats dels operadors**, de major a menor, són

`., [ ]`	accés a tuples i taules
`-, !, *, &`	canvi de signe, negació lògica, desreferència i adreça
`*, /, %`	operadors multiplicatius
`+, -`	operadors additius
`<, >, <=, >=`	operadors relacionals
`==, !=`	igualtat i desigualtat
`&&`	i lògica
`||`	o lògica

  - **ATENCIÓ!** La prioritat de l'operador de desreferència `*` és més petita que la de l'operador d'accés a un camp d'una tupla. Per tant, per accedir a un camp d'un paràmetre de tipus tupla passat per referència cal usar parèntesi `(*nom_param).camp`, o bé l'operador `nom_param->camp`.

    ```
    typedef struct {
        int x,y;
    } punt;
    ...
    void modificaPunt(punt *const p)
    {
        (*p).x = (*p).x + 3;
        p->y = p->y + 3;
    }
    ```

- L'**associativitat** és la mateixa que en la notació algorísmica llevat dels operadors unaris `-` i `!`, en què és de dreta a esquerra.

D.8 Implementació d'accions

acció ::= **acció** nom_acció*(param_formals,* *param_formals, . . .)* *definició_constants* *definició_tipus* *declaració_variables* *sentències* **facció**	*acció ::=* `void` nom_acció(*param_formals,* *param_formals, . . .*) `{` *definició_constants* *definició_tipus* *declaració_variables* *sentències* `}`

param_formals ::=	*param_formals ::=*
ent nom_param, nom_param, ... : nom_tipus	nom_tipus nom_param `const` nom_tipus nom_param
sort nom_param, nom_param, ... : nom_tipus	nom_tipus `*const` nom_param nom_tipus nom_param
ent/sort nom_param, ... : nom_tipus	nom_tipus `*const` nom_param nom_tipus nom_param

D.8.1 Llenguatge C

- Paràmetres formals
  - El primer cas és per als paràmetres d'entrada que no són de tipus taula.
  - El segon cas és per als paràmetres d'entrada de tipus taula.
  - El tercer cas és per als paràmetres de sortida que no són de tipus taula.
  - El quart cas és per als paràmetres de sortida de tipus taula.
  - El cinquè cas és per als paràmetres d'entrada/sortida que no són de tipus taula.
  - El sisè cas és per als paràmetres d'entrada/sortida de tipus taula.
- Us dels paràmetres formals
  - Tot paràmetre passat per referència, els de `*`, s'ha d'usar dins del cos de l'acció com a `*nom_param`. Cal tenir en compte que, si aquest paràmetre es vol tornar a passar per referència a una altra acció, en lloc de `&*nom_param` cal posar `nom_param`.
  - L'accés a un camp d'un paràmetre de tipus tupla passat per referència es pot fer de dues formes: `(*nom_param).nom_camp` o bé `nom_param->nom_camp`.

D.9 Implementació de funcions

funció ::=	*funció ::=*
funció nom_funció(*param_funció*, *param_funció*, ...) **retorna** nom_tipus *definició_constants* *definició_tipus* *declaració_variables* *sentències* **retorna** *expressió* **ffunció**	nom_tipus nom_funció(*param_funció*, *param_funció*, ...) `{` *definició_constants* *definició_tipus* *declaració_variables* *sentències* `return` *expressió*`;` `}`

param_funció ::=	*param_funció ::=*
ent nom_param, nom_param, ... : nom_tipus	nom_tipus nom_param `const` nom_tipus nom_param

D.9.1 Llenguatge C

- Paràmetres formals
  - El primer cas és per als paràmetres d'entrada que no són de tipus taula.
  - El segon cas és per als paràmetres d'entrada de tipus taula.

D.10 Identificadors

- Tots els símbols que comencen per *nom_* són identificadors.
- Tot identificador és una seqüència de lletres, dígits i caràcters subratllada (_) que comença amb una lletra.

- Les lletres majúscules i minúscules es consideren diferents. Per tant, identificadors com *punt* i *Punt* són diferents.

D.11 Valors

D.11.1 Notació algorísmica

- Els valors enters són una seqüència de dígits precedits opcionalment de signe.

```
32        -347        0
```

- Els valors reals s'han d'escriure amb el punt decimal i es poden escriure en notació exponencial.

```
0.0         -3.56e12         48.2E-3
```

- Els valors de tipus caràcter s'escriuen entre cometes senzilles.

```
'A'         'a'         '%'
```

- Els valors booleans són **cert** i **fals**.

D.11.2 Llenguatge C

- La sintàxi en C per als valors enters, reals i caràcter és la mateixa que en la notació algorísmica.
- Els valors booleans són `true` i `false`.
- Es poden usar els valors "especials" de tipus caràcter següents:

`'\n'`	salt de línia
`'\t'`	tabulador
`'\0'`	caràcter `NUL`

D.12 Entrada/sortida

D.12.1 Notació algorísmica

- Es proposen les operacions predefinides d'entrada/sortida següents:

accio *EscriureEnter*(**ent** e : **enter**)
accio *EscriureReal*(**ent** r : **real**)
accio *EscriureCaràcter*(**ent** c : **caracter**)
funcio *LlegirEnter*() **retorna enter**
funcio *LlegirReal*() **retorna real**
funcio *LlegirCaràcter*() **retorna caracter**

D.12.2 Llenguatge C

- Per tal de facilitar l'us de l'entrada/sortida es proposa que s'usin les operacions d'entrada/sortida implementades en el fitxer `entsort.c`. Aquestes operacions són traducció de les usades en la notació algorísmica.

D.13 Cadenes de caràcters

D.13.1 Notació algorísmica

- Són taules de caràcters del tipus

  tipus
  cadena = **taula** [0..10] **de caracter**
  cad = **taula** [0..3] **de caracter**
  pcad = **taula** [0..2] **de caracter**
  ftipus

- Els valors de tipus cadena de caràcters es representen com una seqüència de caràcters entre cometes dobles:

  "*Hola*"

- Els valors de tipus cadena de caràcters són compatibles, pel que fa a l'assignació, amb qualsevol taula de caràcters amb longitud suficient per encabir-los.

- El conveni de llargària emprat és el de sentinella. El sentinella és el caràcter `NUL`. Per exemple, el valor `"Que"` es representa com

0	1	2	3
'Q'	'u'	'e'	`NUL`

- La regla d'assignació és vàlida per al pas de paràmetres d'entrada.

A continuació es mostren alguns exemples d'assignacions de valors de tipus cadena de caràcters.

var
c1 : *cadena*
c2 : *cad*
c3 : *pcad*
fvar
c1 := "*Que*"
{*c1* = 'Q' 'u' 'e' `NUL` ? ? ? ? ? ? ?}
c2 := "*Que*"
{*c2* = 'Q' 'u' 'e' `NUL`}
c3 := "*Que*"
{Assignació invàlida, *c3* no té suficient longitud}

D.13.2 Llenguatge C

- Hem establert, en la notació algorísmica, el mateix conveni per a les cadenes de caràcters que s'utilitza en `C`. L'única diferència fa referència a la regla d'assignació: no és possible assignar un valor cadena a una variable d'algun tipus cadena. Si que és possible, però, passar una constant cadena de caràcters com a paràmetre.

- La llibreria estàndard de `C` proporciona un conjunt de funcions que faciliten el treball amb cadenes de caràcters. Vegeu les especificacions de `<string.h>` a l'apèndix A de [KERN88].

D.14 Subprogrames sense paràmetres

D.14.1 Llenguatge C

- Cal indicar mitjançant la paraula clau `void` a la capçalera que un subprograma no té paràmetres. Per exemple,

– Prototipus:

```
int SenseParametres(void);
```

– Crida:

```
  ...
i = SenseParametres();
  ...
```

– Implementació:

```
int SenseParametres(void)
{
  ...
  return ...;
}
```

Appendix E

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of

it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografia

[ABLA94] Abrahams, Paul W.; Larson, Bruce A.. *UNIX para impacientes*. Addison Wesley, 1988.

[KERN88] Kernighan, Brian W.; Ritchie, Dennis M.. *The C Programming Language*. 2a edició. Prentice-Hall, 1988.

[STAL91] Stallman, R.. *GNUEMACS V18 Reference Manual*. No publicat, 1991.

Índex alfabètic