Introducció a la Informàtica
Grau en Estadística

TREBALL
EN LLENGUATGE R
NOTES DE LABORATORI

ROBERT JOAN ARINYO

Departament de Ciències de la Computació
Universitat Politècnica de Catalunya

Barcelona, Setembre 2019

Aquest document es distribueix
sota una llicència Creative Commons 3.0 de:
Reconeixement
No comercial
Compartir sota la mateixa llicència

Durant el desenvolupament, un programa de computador es llegeix moltíssimes més vegades que no s'executa.

La bellesa del codi

Índex

Capítol 1

Introducció

La informàtica és una ciència que estudia les tècniques d'emmagatzemament i tractament automàtic de la informació. Els dos grans components de la informàtica són: el *hardware* (les màquines) i el *software* (els programes).

1.1 El hardware

El hardware és el conjunt de dispositius físics (materials). L'element central és el *computador* al qual es connecten els elements auxiliars anomenats *perifèrics* tals com les unitats d'emmgatzemament (discos de diverses classes), unitats de comunicació home-màquina (teclats, pantalles), unitats de comunicació màquina-màquina (mòdems), etc.

Un computador és una màquina electrònica capaç d'efectuar un conjunt petit d'operacions simples de càlcul a elevada velocitat i sense intervenció humana.

Avuí dia, la immensa majoria dels computadors existents representen la informació mitjançant valors quantificats en una notació numèrica de base fixada (usualment 2), la informació pren valors en un conjunt finit numerable i només es pot modificar de forma discreta. Com que aquests computadors representen tota la informació mitjançant dígits (números), s'anommenen computadors *digitals*.

Depenent de la forma en la que efectuen els càlculs, els computadors digitals poden ser seqüencials, vectorials o paral.lels. Els seqüencials efectuen a cada instant una única operació; els vectorials també efectuen una única

operació però de caràcter vectorial (per exemple, producte escalar de dos vectors en una sola operació); els computadors paral.lels executen diverses operacions simultàniament.

L'organització genèrica dels computadors digitals, també dita arquitectura, és la coneguda amb el nom d'arquitectura de Von Neumann, el matemàtic que la va inventar. Les unitats principals d'aquesta arquitectura, il.lustrada a la Figura 1.1, són: la unitat central de procès (CPU, *Central Process Unit*), la memòria central (MC, també dita RAM, *Random Access Memory*), i els perifèrics. La CPU consta de dos elements: la unitat de control, (UC, *Control Unit*), que és l'encarregada del control del funcionament del computador, i la unitat aritmètica i lògica, (ALU, *Arithmetic and Logic Unit*) que és l'encarregada d'efectuar els càlculs.

La idea bàsica de l'arquitectura Von Neumann és que la CPU interpreta i tracta només la informació emmagatzemada en la memòria central. La comunicació entre aquests dos elements es fa a través del bus (pot pensar-se que un bus és simplement un cable elèctric compost de molts fils). En els computadors moderns, els perifèrics es comuniquen amb el bus del computador sota el control d'una unitat anomenada *canal*.

Al Capítol 2, tornarem a tractar amb aspectes relacionats amb el hardware només, però, per a explicar breument l'organització lògica de la informació emmagatzemada en un disc. Aleshores podrem justificar la necessitat que l'usuari dissenyi de manera adient com organitza la seva pròpia informació. Aquest extrem resulta de cabdal importància per a tenir èxit en la programació de computadors i en la informàtica en general.

1.2 El software

Un programa de computador és una descripció d'una seqüència d'instruccions o passos de càlcul que defineixen una tasca que realitzarà un computador.

El software és el conjunt de programes de computador, es a dir, conjunts d'instruccions que, d'alguna manera, el computador sabrà interpretar.

1.2.1 Una classificació del software

En general, els elements software es poden classificar dins d'una de les tres families següents: sistemes operatius, programes d'utilitat i programes d'aplicació. La frontera que separa aquestes tipologies és cada dia més difusa.

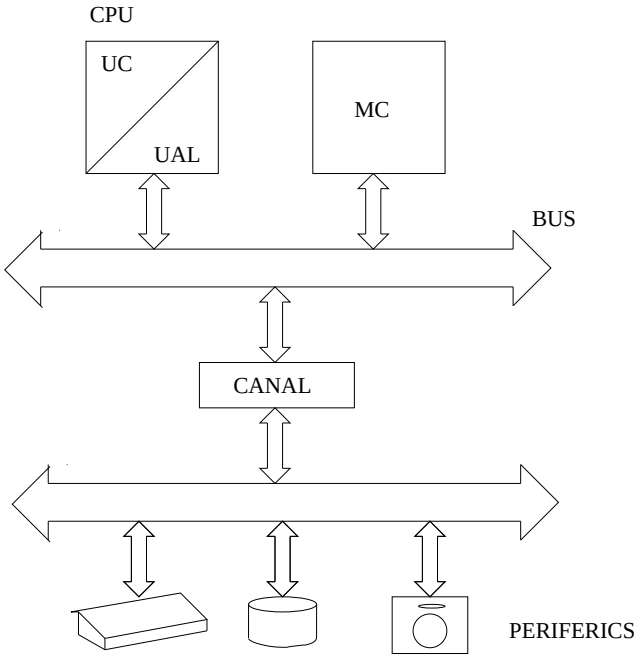

Figura 1.1: Arquitectura de Von Neumann.

El sistema operatiu està compost per un conjunt de programes que controlen la gestió dels recursos del sistema de manera segura i eficaç i, alhora, proporcionen la connexió funcional (la interfase) entre el hardware i l'usuari del computador.

Les utilitats són programes que permeten el desenvolupament d'altres programes. Entre les més importants estan els editors de fitxers i els compiladors. Els editors permeten usar un computador com una mena de màquina d'escriure. Els compiladors permeten traduir allò que les persones codifiquen en una representació més adient pels computadors digitals.

Les aplicacions són els programes desenvolupats per resoldre algun problema concret amb l'ajut del computador.

L'eina amb la qual es representen els programes de computador són els llenguatges de programació. Un llenguatge de programació no és res més que una definició rigorosa d'una sintaxi i una semàntica associada a la sintaxi. La sintaxi defineix la manera en la que cal codificar les instruccions que descriuen la solució d'un problema (Barcelona s'escriu amb B i no amb V). La semàntica defineix el significat de la sintaxi (7 * 8 significa el producte del valor set pel valor vuit).

De les classificacions que es poden fer dels llenguatges de programació, la més útil per a nosaltres és la que els classifica com: llenguatges de màquina, llenguatges simbòlics de baix nivell i llenguatges simbòlics d'alt nivell. Els llenguatges de màquina són propis de cada computador i, antigament, només es podien codificar mitjançant representacions binàries de zeros i uns. Podem dir que aquests són els llenguatges que realment poden interpretar els computadors digitals.

Els llenguatges de baix nivell simbòlics, també coneguts com ensambladors (*assembly* en anglès) són distints per a cada computador o familia de computadors, ara bé, tenen representació simbòlica. Això vol dir que usen símbols mnemònics per a representar conceptes.

Els llenguatges d'alt nivell són simbòlics i, en principi, cadascun d'ells té una sintaxi i una semàntica universals (estàndar); és a dir, són iguals per a qualsevol computador. La codificació és molt propera a la manera usual que tenim els humans de representar els processos de càlcul. Exemples ben coneguts d'aquests llenguatges són el `Pyhton`, `FORTRAN`, `Pascal`, `C`, `C++`, `Java` i `R`.

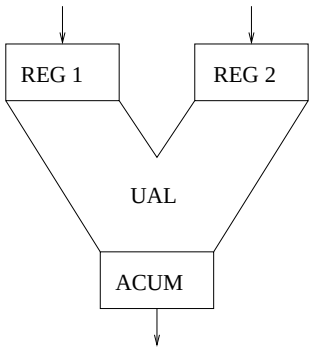

Figura 1.2: Unitat aritmètica i lògica.

1.2.2 Un exemple de nivells de llenguatge

La Figura 1.2 il.lustra una unitat aritmètica i lògica de tres registres d'un hipotètic computador de Von Neumann. Els registres REG1 i REG2 són unitats electròniques que emmagatzemen les dades sobre les quals cal efectuar alguna operació. El regsitre ACUM, anomenat *acumulador*, és el registre on resta emmagatzemat el resultat de la darrera operació efectuada sobre les informacions emmagatzemades en els registres.

Com que suposem que el computador és de Von Neumann, abans d'efectuar una operació cal copiar als registres la informació de les dades a operar que estan a la memòria central. Un cop efectuada l'operació entre les dades dels registres, cal copiar la informació de l'acumulador, és a dir, el resultat, a la memòria central.

Suposem el programa codificat en llenguatge **R**

```
x <- y + z
```

la semàntica del qual és: suma les informacions representades per les variables `y` i `z` i fes que el resultat sigui representat per la variable `x`. Aleshores, en un hipotètic llenguatge ensamblador simbòlic de baix nivell, aquest segment de programa podria representar-se com

```
X OCT*4
Y OCT*4
Z OCT*4
LOAD R1, @Y
```

```
LOAD R2, @Z
ADD
STORE @X
```

On les tres primeres línies defineixen la quantitat de memòria central que cal reservar per emmagatzemar la informació de les variables. La línia `LOAD R1, @Y` indica que cal copiar al registre `R1` la informació representada per la variable `Y` a la memòria central. El significat de la línia `LOAD R2, @Z` és anàleg. La línia `ADD` dona l'ordre de que s'efectui l'addició de les informacions carregades als registres. La línia `STORE @X` copía a la posició de memòria indicada per `X` el resultat de l'operació el qual està emmagatzemat a l'acumulador.

Finalment, suposem que l'operació `LOAD` es representa pel conjunt de bits `0101`, `ADD` es representa per `1000` i `STORE` es representa per `1100`. Suposem també que el registre `R1` es representa per `001` i que `010` representa el registre `R2`. Finalment, suposem que la variable `X` està a la posició `1000` de la memòria central, la variable `Y` està a la `1010` i la variable `Z` està a la `1100`. Aleshores, en aquest hipotètic llenguatge màquina, el mateix segment de programa de més amunt es podria representar en binari sobre instruccions de setze bits, com

```
0101 001 000001010
0101 010 000001100
1000 000 000000000
1100 000 000001000
```

Notis que la representació de les variables ha desaparegut. Les variables ara no juguen cap funció, atès que la informació està a la memòria central del computador.

Recomanem l'estudiant que, usant el llenguatge ensamblador simbòlic de baix nivell i el llenguatge màquina indicats, intenti escriure els programes corresponents als segments de programa en llenguatge **R** següents

```
x <- x + y          x <- x + y + z
```

Si s'interpreten els programes vistos seguint la sintàxi d'**R**, s'ha trobat cap aspecte en aquesta interpretació que xoqui amb la interpretació usual de les variables que hom fa en l'àlgebra?

1.3 Enginyeria del software

El procés de dissenyar, implementar i mantenir solucions software eficaces per problemes complexos s'anomena *enginyeria de la programació* o *enginyeria del software*. El procés de construcció d'una solució software completa rep el nom de *cicle de vida del software* i, en general, es considera compost pels següents passos:

- Anàlisi de requeriments: determina el conjunt de requeriments que cal que satisfaci el software, és a dir, defineix l'aplicació o problema a resoldre.
- Especificació: es descriuen formalment les entrades (dades), les sortides (resultats) i els processos que generen les sortides a partir de les entrades.
- Disseny: s'identifiquen els principals tipus de dades i operacions que cal aplicar-los-hi així com els mòduls funcionals (solucions dels subproblemes) que componen l'aplicació.
- Implementació: també anomenat *programació*, en el que es codifiquen els algorismes usant un llenguatge de programació de computadors.
- Instal.lació i prova: s'instal.la el programa resultant en el computador corresponent i es verifiquen la correctesa i adequació a les especificacions.
- Manteniment: un cop un programa està en explotació, el manteniment avalua el rendiment del programa, detecta i corregeix possibles errors existents i diagnostica la necessitat de possibles actualitzacions.

1.4 Tòpics de la informàtica

La informàtica

- No és únicament programar. La programació (objectiu d'aquest curs) només és el mitjà d'expressió bàsic de la informàtica.
- No consisteix en la simple utilització d'un computador en un ambient qualsevol, per exemple, usar un full de càlcul o un processador de

textos, navegar pel *World Wide Web* a Internet o rebre i trametre missatges per correu electrònic.

- No és una subdisciplina de l'enginyeria electrònica ni de les matemàtiques, malgrat estar fortament relacionada amb totes dues.

La informàtica és

- L'estudi dels algorismes i els computadors: les teories, models abstractes de dades, realització mecànica, fiabilitat, verificació i mesura de la seva eficàcia.
- Una disciplina matemàtica perquè les teories i l'estil de recerca i comunicació són necessàriament rigorosos i segueixen pautes matemàtiques.
- Una disciplina científica perquè les teories i abstraccions s'avaluen i es proven en el laboratori de manera experimental.
- Una disciplina d'enginyeria perquè aplica el disseny i l'anàlisi per a obtenir sistemes eficaços, fiables i econòmicament factibles que resolen problemes tècnics d'interès per a la societat.

Capítol 2

Organització del disc

Tot seguit farem una descripció de com s'organitza la informació emmgatazemada en un disc. La descripció es basa en la tècnica usada en la tecnología tradicional de disc dur magnètic. Si bé en els discs basats en noves tecnologies, tals com la memoria de bombolla, l'organització no té perquè ser realment la mateixa, el model que descrivim s'aplica igualment.

2.1 Sistemes operatius basats en disc

En els anys 50 del segle XX, els computadors portaven instal.lats a la part frontal un conjunt d'interruptors elèctrics a través dels quals, un cop el computador tenia tensió elèctrica, els operadors l'engegaven tot entrant manualment amb els interruptors un programa de càrrega inicial que es coneixia com el *bootstrap*.

Com que en tallar el subministrament de corrent al computador totes les dades en memòria central es perden, aquesta operació, més aviat feixuga i molesta, (coneguda com *booting, boot* o catalanitzat *botar*), calia repetir-la cada cop que s'encenia el computador. Paral.lelament, en no haver elements d'emmagatzematge, els programes, dades i resultats dels càlculs calia emmagatzemar-los en estris tals com cintes de paper i targes perforades.

L'aparició i desenvolupament dels discs magnètics que permetien l'emmagatzematge permanent d'informació va comportar un canvi radical en la manera d'explotar els computadors electrònics. Primer, l'operació de *botar* un computador es va poder fer automàticament.

Els computadors incorporaren una petita memòria no volàtil que contè un petit *bootstrap* el qual s'executa en connectar la màquina al subministrament de corrent elèctric. Aquest petit programa té per missió transferir des del disc dur a la memòria central tots els programes inicials que calen per fer funcionar el computador. A més, la presència del disc va fer possible que els programes dels usuaris així com les dades i resultats foren emmagatzemats de manera permament en el propi disc, fent innecessàries les cintes de paper, targes peforades i d'altres tecnologies poc eficients. El que acabem de fer és descriure de manera somera com funciona un sistema operatiu basta en disc. Per tant,

> Un sistema operatiu es diu que està basat en disc si existeix una unitat d'emmagatzematge permanent que permet 1) fer el *boot* automàtic del computador i 2) qualsevol operació de transferència d'informació entre el computador i aquesta unitat.

Noteu que la unitat d'emmagatzematge no té perquè ser un disc estrictament, sinó que allò que importa és la funcionalitat que ofereix.

2.2 Organització física

Tot i que en general l'organització física de la informació en un disc pot ser més complexa, ací descriurem els elements bàsics, suficients per un curs introductori. De fet, com que no es tractarà de la programació a baix nivell, al curs no caldrà usar aquests conceptes. Els descrivim només per completesa de l'explicació.

Des d'un punt de vista físic considerem que la informació en els discs s'estructura segons dos conceptes: *pistes* i *sectors*.

Una pista és una corona circular del disc d'una amplada prefixada. Un sector és la intersecció d'una pista amb un sector circular del disc. La Figura 2.1 mostra esquemàticament un disc amb aquesta estructura.

En general, la unitat d'informació que es transfereix entre la memòria central i el disc en una operació de lectura/gravació és la que pot emmagatzemar un sector.

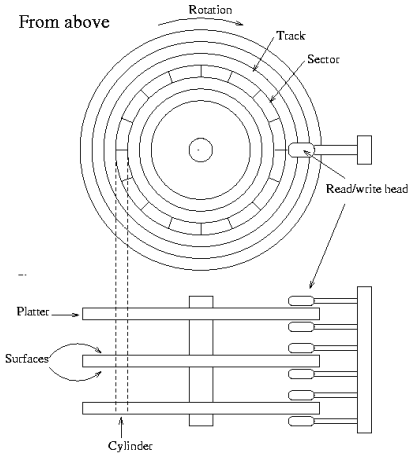

Figura 2.1: Organització d'un disc en pistes i sectors (Font: Vikipedia).

2.3 Concepte de fitxer

Com ha estat dit més amunt, llevat que hom treballi en el desenvolupament de sistemes operatius, els usuaris no tenen perquè preocupar-se de l'estructuració física de la informació en els discs. Normalment el disc s'utilitza a un nivell conceptual abstracte molt elevat basat en el concepte de *fitxer*.

Un *fitxer* (*file* en anglès) és una unitat lògica que emmagatzema informació, físicament disposada sobre un suport secundari d'informació permanent i que s'identifica amb un nom únic. Per més que darrerament s'ha fet popular la denominació de *carpeta* (de l'anglès *folder*) ací emprarem el mot clàssic de fitxer.

Hi ha dues classes de fitxers: *fitxers planers* o simplement *fitxers* i *directoris*. Els fitxers emmagatzemen informació que anomenarem final. Per exemple un programa de computador, les dades per un programa, els resultats que ha generat l'execució d'un programa, el text d'una carta, el fitxer que contè aquest escrit, etc.

Un directori és un fitxer tal que pot contenir fitxers planers i d'altres directoris. És l'eina que s'utilitza per definir la jerarquia de fitxers d'un sistema basat en disc.

En general la informació que emmagatzemi un fitxer o directori ocuparà diversos sectors els quals podran estar escampats per diverses pistes del disc.

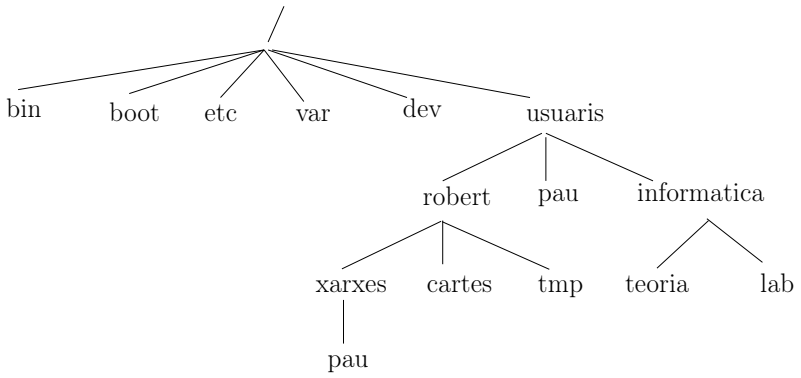

Figura 2.2: Exemple de jerarquia de fitxers.

Però això a l'usuari no l'ha de preocupar, el sistema operatiu se n'ocupa de gestionar la transferència i emmagatzematge.

2.4 La jerarquia de fitxers

En els sistemes operatius basats en disc, els fitxers s'organitzen segons una estructura jeràrquica de directoris. L'origen de la jerarquia és un directori que s'anomena directori *arrel* i que tradicionalment es designa amb el símbol / en els sistemes Linux i amb el símbol \ en d'altres. De moment usarem el símbol /.

Assumirem que a la jerarquia hi ha un directori de nom `/usuaris` (de vegades el nom és `/home`) del qual depenen la resta de directoris dels usuaris. Així, si hi ha un usuari de nom `robert`, hi haurà un directori `/home/robert` a partir del qual es situaran tots els seus fitxers. Aquest directori s'anomena directori `casa` de l'usuari i acostuma a representar-se per la parella de símbols ~/ (titlla i /).

Un exemple de jerarquia de directoris pot ser l'indicat a la Figura 2.2. En ella, el directori arrel contè sis fitxers identificats com `bin, boot, etc, var, dev` i `usuaris`. Aquest darrer és un directori que contè tres fitxers, `robert, pau` i `informatica`. (Nota que no hi ha accents. No és bona pràctica usar identificadors que tinguin símbols que no pertanyin a l'alfabet anglès). Finalment, el primer dels tres directoris contè tres fitxers més: `xarxes, cartes` i `tmp`.

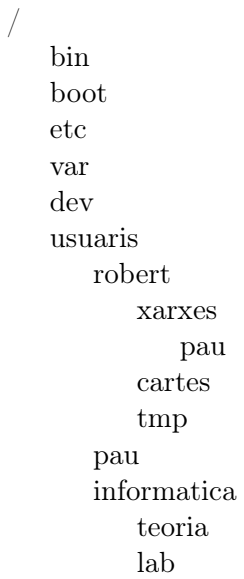

Figura 2.3: Jerarquia de fitxers de la Figura 2.3 representada de manera diferent.

Per referir-se a un directori que està dins d'un altre directori s'usa la denominació de *subdirectori*. A l'exemple de la Figura 2.2, el directori `tmp` és un subdirectori del directori `robert` i aquest al seu torn és un subdirectori del directori `usuaris`. Simètricament, un directori que en conté un altre és diu que és el *pare* del directori contingut. Per exemple, el directori `robert` és el pare del directori `tmp`.

Hi ha sistemes operatiu que mostren la jerarquia de directoris segons la forma dibuixada a la Figura 2.3. Noteu que són formes de representació equivalents.

2.5 Nom d'un fitxer

Per a que el sistema de fitxers sigui útil, cal que no hi hagi ambigüitats a l'hora d'identificar un fitxer. Això requereix que cada fitxer tingui un únic nom i que cada nom correspongui a un únic fitxer. Aleshores, a la Figura 2.2 no hi ha dos directoris de nom `pau`?

El nom d'un fitxer és una cadena de caràcters tal que el primer és el símbol del directori arrel, /, seguit per els successius noms de subdirectoris que defineixen el camí dins la jerarquia fins a arribar al pare del fitxer considerat i, finalment, el nom del propi fitxer considerat, separats pel símbol /. Ara resulta clar que a l'estructura de fitxers de la Figura 2.2 no hi ha dos fitxers que tinguin per nom `pau`. Els noms dels fitxers realment són

```
/usuaris/robert/xarxes/pau
```

i

```
/usuaris/pau
```

Del nom complet d'un fitxer de vegades s'en diu nom *absolut*.

Si bé els caràcters que poden usar-se per definir noms de fitxers són pràcticament tots, la bona pràtica ensenya que resulta aconsellable usar només caràcters del conjunt format per les lletres de l'alfabet anglès (res de lletres accentuades), dígits àrabs, el guió (-) i el guió baix (_)). A més, cal evitar aquells noms als que el sistema operatiu pugui assignar un significat específic. Per exemple, es recomana que, en general, el primer caràcter sigui alfanumèric (lletres o dígits àrabs). Una pràctica molt estesa consisteix a definir noms de fitxer deixant espais en blanc. Aquesta pràctica és poc recomanable perquè pot generar problemes inesperats, especialment quan els fitxers es traspassen a un altre sistema operatiu. Per exemple, un nom de fitxer tal com

```
Carta a la Mireia de setembre 2015
```

fora millor canviar-lo per

```
cartaAlaMireia0915
```

de manera que la primera lletra sigui minúscula i les majúscules s'empren per a fer la composició. Un nom millor encara fora

```
carta_mireia_0915
```

on totes les lletres són minúscules, per estalviar haver de prèmer la tecla de canvi de mode, i la composició es fa amb guionet baix. No és simplement una qüestió de gustos, que també ho és, sinó d'plicar una mnemotècnia sistemàtica, eficaç i pràctica.

Una qüestió addicional és la que fa refència a les extensions. En programació de computadors, és una pràctica habitual que els noms del fitxers tinguen dues components separades per un punt, per exemple

```
carta_mireia_0915.txt
```

on l'extensió és la cadena de caràcters `.txt`. En sistemes operatius avançats i fiables com el Unix/Linux, l'extensió no té cap altre valor que el mnemònic. Al darrer exemple, l'extensió `.txt` podria voler indicar que el fitxer conté el text d'una carta. Però en els sistemes operatius avançats, un fitxer de nom

```
carta_mireia_0915.exe
```

podria tenir exactament el mateix contingut i no implicaria cap altra propietat del fitxer.

D'altres sistemes operatius, l'extensió defineix algunes de les propietats que suposadament s'atribueixen al fitxer de manera que l'extensió `.txt` cal associar-la a fitxers que contenen text mentre que `.exe` s'associaria a un programa o *script* que es pot executar. Hi ha extensions que venen imposades no pel sistema operatiu sinó pels programes d'utilitat que els maneguen. Per exemple, l'aplicació `xpdf` de Unix/Linux espera que els fitxers siguin codificats en Portable Document Format (PDF) i per tant que els noms dels fitxers segueixen el format `nom.pdf`. Val a dir que l'aplicació `xpdf` tracta correctament qualsevol fitxer en format PDF encara que el nom no segueixi el conveni.

2.6 El directori de treball

Resulta aparent que, si cada cop que un usuari ha de referenciar un fitxer cal que doni el nom complet, referenciar fitxers resultarà una feina feixuga. La majoria de sistemes operatius basats en disc resolen aquest problema potencial d'una manera simple mitjançant el concepte de *directori de treball*.

El directori de treball és aquell directori on resten reflectides totes les interaccions de l'usuari amb el computador i des d'on s'efectuen totes les operacions de transferència d'informació entre el computador i el disc en tots dos sentits. És responsabilitat de l'usuari definir en tot moment quin cal que sigui aquest directori. Aleshores, els noms dels fitxers es defineixen en relació al directori de treball, és a dir, els noms dels fitxers comencen amb el nom del directori de treball. Per exemple, considerem a la Figura 2.2 el fitxer que té per nom absolut

```
/usuaris/robert/xarxes/pau
```

o el que, per l'usuari `robert`, és el mateix

```
~/xarxes/pau
```

Si definim com a directori de treball

```
/usuaris/robert
```

aleshores el fitxer considerat té per nom

```
xarxes/pau
```

mentre que si el directori de treball és

```
/usuaris/robert/xarxes
```

aleshores el nom del mateix fitxer és simplement

```
pau
```

El nom d'un fitxer respecte d'un directori de treball és diu que és un *nom relatiu*. Si el directori de treball és exactament el pare d'un fitxer, el nom d'aquest fitxer és el nom *local*, com ha estat el cas del darrer exemple del fitxer de nom `pau`. El conjunt de noms de directori que identifiquen els directoris des de l'arrel fins al pare del fitxer considerat rep el nom de *camí* (*path* en anglès).

2.7 Un exemple d'organització

Prendrem com a exemple la informació que un estudiant d'Introducció a la Informàtica del Grau d'Estadística.

Per no fer-ho massa llarg, suposem que un estudiant, de nom Robert, està matriculat de les assignatures d'Informàtica, Àlgebra i Economia i que en totes elles el computador resulta una eina imprescindible, bé per desenvolupar programes de computador, el cas d'Introducció a la Informàtica, bé com a magatzem d'informacions corresponents a apunts, llibres, exercicis i documents diversos d'informació administrativa del curs. Suposant que el directori casa de l'usuari `robert` és

```
/usuaris/robert
```

Una organització convenient pel seu disc en tot allò que fa referència a l'escola on estudia podria ser com la següent.

```
robert/escola/
robert/escola/matricula
robert/escola/assignatures/informatica
robert/escola/assignatures/algebra
```

```
robert/escola/assignatures/economia
...
robert/escola/assignatures/informatica/documents
robert/escola/assignatures/informatica/apunts
robert/escola/assignatures/informatica/lab
robert/escola/assignatures/informatica/lab/sessio_1
robert/escola/assignatures/informatica/lab/sessio_2
...
robert/escola/assignatures/informatica/lab/sessio_n
...
robert/escola/assignatures/informatica/apunts/elLlibre.pdf
robert/escola/assignatures/informatica/apunts/treballEnR.pdf
...
robert/escola/assignatures/informatica/lab/sessio_1/hello.R
robert/escola/assignatures/informatica/lab/sessio_1/prodEscalar.R
```

Pel que fa a una assignatura com Economia, on el computador probablement és bàsicament una eina d'emmagatzemament, l'organització podria ser

```
robert/escola/assignatures/economia
robert/escola/assignatures/economia/documents
robert/escola/assignatures/economia/web
robert/escola/assignatures/economia/apunts
robert/escola/assignatures/economia/exercicis
robert/escola/assignatures/economia/exercicis/tema_1
robert/escola/assignatures/economia/exercicis/tema_2
...
robert/escola/assignatures/economia/exercicis/tema_n
```

Tot això no és res més que un suggeriment. Cada usuari pot definir l'organització del seu disc de la manera que millor li plagui. En qualsevol cas, una organització de la informació sense lògica és un clar impediment per a obtenir un bon rendiment de la informàtica.

Capítol 3

Processament de programes

Al Capítol 1 hem vist que, per una banda, un llenguatge d'alt nivell és el mateix per a tots els computadors i que, per altra banda, cada computador realment només pot interpretar el seu propi llenguatge màquina. Aleshores, serà possible construir programes codificats en llenguatge d'alt nivell i que siguin executats per qualsevol computador? La resposta és, en principi, afirmativa i la solució rau en aplicar una mena de procés de traducció.

La traducció d'un programa expressat en un llenguatge d'alt nivell a un expressat en llenguatge màquina es fa actualment segons dues tècniques diferents. Una s'anomena *compilació* i l'altra s'anomena *interpretació*.

3.1 Processament amb compilació

La tècnica de compilació aplica un procés que pot ser descompost en tres fases:

1. Fase de compilació: el codi del programa escrit segons la sintaxi d'un llenguatge d'alt nivell es tradueix al llenguatge màquina corresponent a un computador concret. El resultat s'anomena *programa objecte* i el procés de traducció l'efectua un programa d'utilitat anomenat *compilador*.

2. Fase de muntatge: el codi màquina resultant de la fase de compilació es completa amb codi addicional previament desenvolupat i emmagatzemat en les anomenades llibreries del llenguatge. El procés l'efectua un

programa d'utilitat anomenat *muntador* i el resultat és un programa que es diu *carregable*.

3. Fase de càrrega i execució: el programa carregable es transforma de manera que pugui ser posicionat en un lloc concret de la memòria central del computador. El resultat és el programa *executable*. Finalment, el programa executable es disposa de manera efectiva en la memòria central i s'inicia l'execució.

Val a dir que, avuí dia, els sistemes operatius disponibles fan que no necessàriament calgui passar de manera explícita per totes i cadascuna de les fases indicades. En general, el sistemes operatius moderns executen les fases 1 i 2 automàticament sempre que no apareguin errors.

Una altra qüestió important és la relacionada amb els errors. Cadascuna de les fases 1 i 2 pot acabar de dues maneres diferents mentre que la fase 3 pot acabar de tres maneres. En general, si una fase no conté errors, es passa a la fase següent. La fase de compilació pot donar lloc a errors de sintaxi, és a dir, errors que signifiquen que hi ha alguna part que no ha estat representada segons el que especifica la sintaxi del llenguatge de programació emprat. Corregir aquests errors es usualment senzill perquè el compilador genera els missatges i indicacions oportunes.

La fase de muntatge dona lloc a errors que normalment responen al fet que el programa executable no ha estat convenientment completat, per exemple perquè alguna part del codi que calia incorporar no ha estat trobada a les llibreries. Generalment és un problema que es solventa fàcilment perquè el muntador també genera els missatges que cal.

La fase de càrrega no la considerem perquè, si el muntatge ha estat correcte, ací no han d'haver gaire problemes. Per contra, la fase d'execució és la més problemàtica. Poden apareixer errors d'execució, típicament intentar efectuar una divisió per zero o que el programa intenta accedir a una part de la memòria central a la que no li està permés accedir-hi. Aquests problemes es detecten de manera fàcil perquè el sistema genera els missatges d'error adients.

L'execució pot acabar generant resultats correctes, la qual cosa ens omplirà de satisfacció. Ara bé, l'execució també pot acabar generant resultats incorrectes, cosa que no sempre resulta fàcil de detectar. En aquest cas l'aplicació del programa desenvolupat pot generar una catàstrofe. Cerqueu a la web que va passar amb la nau espacial nordamericana *Mars Path Finder*

el 26 de juny de 2003. Tornarem a comentar la situació al Capítol 7 que tractarà de la metodologia d'anàlisi de problemes.

3.2 Processament amb interpretació

Quan un programa escrit en un llenguatge d'alt nivell es processa seguint la tècnica d'interpretació, la traducció a llenguatge màquina la fa un programa anomenat *intèrpret*. La manera de treballar d'un intèrpret es pot descriure, de manera simplificada, dient que considera a cada instant una única instrucció o pas de càlcul del programa d'alt nivell, la tradueix a la representació de baix nivell i, tot seguit, l'executa. Notis la diferència respecte del cas de treball en compilación on no es desencadenava l'execució fins que no es completaven els pasos previs.

Malgrat la diferència indicada, els errors que poden aparèixer ara són, essencialment, els mateixos que apareixien en el cas del procés amb compilació. Quan l'intèrpret decodifica una instrucció, detecta i indica els possibles errors de tipus sintàctic que hi pugui haver. Un cop corregits els erros de sintaxi, l'intèrpret pot detectar que hi ha elements software externs que caldria que fossin definits i no ho estan. Naturalment l'intèrpret genera el missatge d'error corresponent. Finalment també es poden produir errors d'execució dels quals l'intèrpret donarà la informació oportuna.

3.3 El llenguatge R

R és un llenguatge de programació que pertany a la familia dels llenguatges simbòlics d'alt nivell interpretats. El llenguatge permet incorporar una àmplia gamma de paquets d'anàlisi estadístic i gràfic desenvolupats per la comunitat d'usuaris d'**R**. Això el fa especialment útil per a programar solucions en aquesta disciplina.

Una característica destacable és que **R** es distribueix sota llicència de software lliure i gratuït segons la *Free Software Foundation's General Public License*.

Un concepte bàsic que cal entendre bé és el d'*espai de treball*, (*working space*, en anglès). Aquest concepte fa referència a l'entorn en el qual **R** està treballant a cada moment. Inclou el conjunt d'objectes que hagi definit l'usuari: variables, vectors, matrius, llistes, funcions, etc. Al final de la

sessió de treball en **R**, l'usuari pot emmagatzemar una imatge del seu espai de treball actual el qual serà carregat automàticament la propera vegada que inicii **R**.

3.4 Formes de treballar amb R

Existeixen diverses maneres de treballar en **R** entre les quals trobem: directament en l'intèrpret **R**, en un entorn de programació o en mode treball en lots. Tot seguit els descrivim somerament.

3.4.1 Treball directe

En el treball directe, l'usuari interacciona amb el llenguatge **R** sense cap element intermediari, només s'usen les eines que proporciona el propi llenguatge. La interacció directa es podrà fer en mode intèrpret, és a dir, com si **R** fos una calculadora de butxaca, o en mode comanda (també coneguda com mode *script*). Com veurem en els Capitols 5 i 6, existeixen petites diferències en la forma de treballar depenent del sistema operatiu on s'executa **R**.

3.4.2 Treball en entorns

Un Entorn Integrat de Desenvolupament (*Integrated Development Environment (IDE)* en anglès) és un programa d'aplicació que incorpora diverses eines utilitzades rutinàriament en el desenvolupament de programes especialment en ambients professionals. Entre les eines que inclou acostuma a haver-hi: un editor de fitxers, generació automàtica de codis executables i elements per a la depuració d'errors de programació (*debugging* en anglès). Els IDE més moderns inclouen una interfase gràfica per a la interacció usuari/computador.

Existeixen diversos IDE que permeten treballar en **R** cadascun dels quals té avantatges i inconvenients. Un dels més estesos és el conegut com `RStudio`, que és un IDE amb llicència de softawre lliure i gratuït.

Per a treballar en `RStudio`, només cal desencadenar-lo i usar les diverses funcionalitats que ofereix, tant d'edició de fitxers com de control d'execució. A la Figura 3.1 es mostra la interfície d'`RStudio` executada en Linux/Ubuntu.

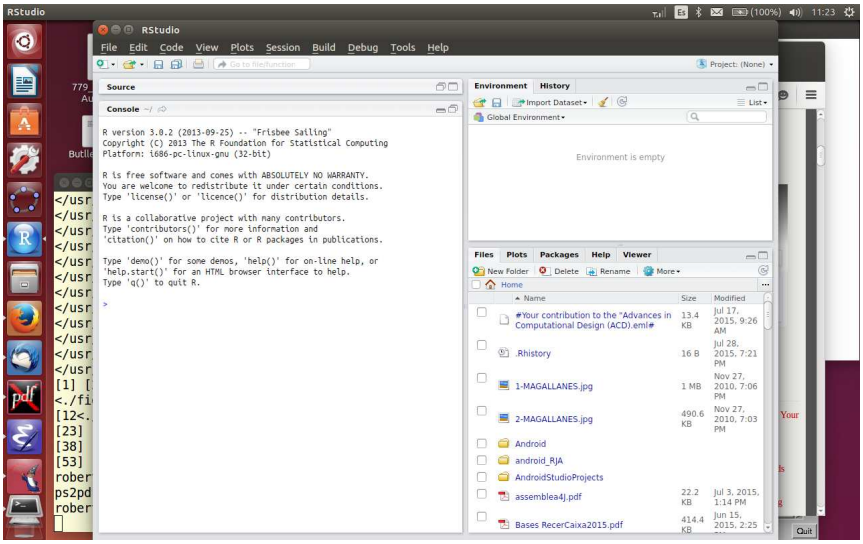

Figura 3.1: Interfase d'usuari d'`RStudio` executada en Linux/Ubuntu.

A la web hom pot trobar nombroses tutories que expliquen com instal.lar-lo i com treballar en ell.

3.4.3 Treball en lots

En programació de computadors, el *treball en lots* (en anglés *batch processing*) és la tècnica usada per executar una seqüència de programes sense intervenció manual. La tècnica es basa en escriure un programa o *script* de control que gestiona el desencadenament de les successives execucions de programes emmagatzemats en el disc. Les dades pels programes en execució s'agafen de fitxers del propi disc i els resultats s'emmagatzemen en d'altres fitxers del mateix disc. La comanda bàsica per a l'execució d'**R** en lots segueix la sintaxi

```
R CMD BATCH [options] nomFitxer.R [fitxersResultats]
```

on

options són opcions que permeten definir la manera exacta d'efectuar el procès per lots.

`nomFitxer.R` és el fitxer **R** que cal executar

`fitxerResultats` és el fitxer on s'emmagatzemaran els resultats.

A la web han estat publicades diverses tutories que expliquen com treballar en lots usant el llenguatge **R**. Com ha estat dit més amunt, cal tenir en compte que aquest mode de treball és útil especialment en explotació d'aplicacions amb tractament de grans volums de dades.

Capítol 4

Instal.lació d'R

Primer descriurem com s'instal.la l'intèrpret bàsic d'**R** en Windows 8 i tot seguit en Linux.

4.1 Instal.lació en Windows

La instal.lació en Windows 8 resulta molt senzilla. El procediment a seguir és:

1. Amb el navegador visita la seu web amb URL

   `https://cran.r-project.org/bin/windows/base/`

   i descarrega el fitxer de l'intèrpret fent clic a l'enllaç mostrat a la primera línia de la pàgina web

   `Download R 3.2.1 for Windows (62 megabytes, 32/64 bit)`

   Es descarregarà un fitxer amb nom `R-3.2.1-win.exe`.

   Els dígits 3.2.1 defineixen la versió de l'intèrpret. Si el que es troba és diferent no importa, segurament serà una versió més nova, per exemple, 3.2.4. Millor, fes la descàrrega.

   Si no us agrada o teniu dificultats en visitar la seu web indicada, demaneu al vostre navegador que faci una cerca amb paraules clau

   `language R Windows download`

   i trobareu diversos llocs des d'on podreu fer la descàrrega.

2. Amb el navegador de fitxers, visita el directori on ha estat descarregat el fitxer, possiblemt `Downloads`, i fes doble clic sobre la icona corresponent. Automàticament s'obriarà l'assistent d'instal.alció. Aconsellem fer una instal.lació estàndard, és a dir, accepteu tots els suggeriments de l'assistent. Per agilitar el treball posterior accepteu que l'assistent inclogui una icona en el vostre escriptori.

Això és tot, l'intèrpret **R** està disponible!

4.2 Instal.lació en Linux

La instal.lació més simple es pot fer a partir del repositori estàndar amb les comandes

```
$ sudo apt-get update
$ sudo apt-get install r-base
```

o bé

```
$ sudo apt-get update
$ sudo apt-get install r-base r-base-dev
```

la qual instal.larà **R**, paquets recomanats així com d'altres fitxers que caldrà per tal d'instal.lar altres paquets addicionals. Malauradament tant l'intèrpret **R** com els paquets disponibles en el repositori no estan sempre al corrent de les actualitzacions més recents.

El projecte `CRAN` dona tota la informació que cal per instal.lar les revisions d'**R** més recents en diversos projectes Linux a la URL

`https://cran.r-project.org/`

Per fer la instal.lació cal aplicar el següent seguit de comandes

1. Inclou a `/etc/apt/sources.list` una entrada pel repositori des del qual es farà la descàrrega, per exemple el de la Universitat de Münster (Alemanya)

   `http://cran.uni-muenster.de/`

Si hom treballa en Ubuntu, a `/etc/apt/sources.list` inclourà la línia

```
deb http://cran.uni-muenster.de/bin/linux/ubuntu XXXX/
```

on `XXXX` representa la versió d'Ubuntu instal.lada: `precise, trusty, utopic, vivid` o d'altres més modernes.

Si hom treballa en Debian la línia a incloure en la llista de repositoris és

```
deb http://cran.uni-muenster.de/bin/linux/debian XXXX/
```

Ara `XXXX` caldrà substituir-ho per un de: `squeeze-cran3, wheezy-cran3, jessie-cran3`.

2. Ara fes la descàrrega i instal.lació amb les comandes estàndard

```
$ sudo apt-get update
$ sudo apt-get install r-base r-base-dev
```

4.3 Altres sistemes operatius

Si el sistema operatiu instal.lat no és ni Windows ni Linux, demaneu al vostre navegador que faci una cerca amb una llista de paraules clau que inclogui, si més no, les següents

```
R language nom-del-vostre-SO download
```

De ben segur que trobareu una munió de webs i tutories que us explicaran com descarregar i instal.lar **R** en el vostre computador.

Capítol 5

Treball en Windows

Descriurem com es treballa en el sistema operatiu Windows 8 i programació en llenguatge **R**. Primer descriurem el treball en mode intèrpret i tot seguit en mode comanda.

5.1 Preliminars

La situació usual quan hom treballa en Windows és treballar en un entorn de finestra gràfica pròpi del sistema operatiu. Per això, si més no en un curs introductori, no resulta especialment interessant utilitzar un IDE, descrit a la Secció 3.4.2.

En Windows també és possible treballar sobre un terminal d'interacció orientada a comanda de caràcters. Només cal iniciar la sessió de treball obrint un terminal de comandes amb la seqüència

Panell de quatre finestres
Executar
cmd

A partir d'ací, la interacció es podria fer via el terminal acabat d'obrir. Aquest mètode detreball té l'inconvenient d'haver d'usar un editor de fitxers extern a **R** per a la creació i edició dels programes. La versió de l'intèrpret d'**R** per Windows ofereix un editor de fitxers integrat molt senzill. Per tant, en tot el que resta d'aquest capítol, suposarem que sempre es treballarà amb les eines del pròpi intèrpret **R** sense usar el terminal. També suposem que

a l'escritori hi ha la icona de l'intèrpret per a poder executar-lo de forma immediata. Si no hi ha la icona d'**R** a l'escriptori o a la barra de tasques, **R** es pot desencadenar amb la seqüència

Panell de quatre finestres
Tots els programes
R

En aquest cas, depenent de la instal.lació que s'hagi fet i de si la CPU del vostre computador és de 32 o de 64 bits, hom pot trobat que hi hagi dos programes executables d'**R**: un amb nom semblant a `Ri386 3.2.1` i un altre amb nom semblant a `Rx64 3.2.1`. Executeu el que vulgueu però, en principi, no hi ha cap necessitat d'usar el segon, que usa més recursos.

5.2 Treball en mode intèrpret

El treball en sistema operatiu Windows 8 i llenguatge de programació **R** en mode intèrpret, consta de tres pasos:

1. Inicialització de l'intèrpret **R** a partir de la icona a l'escriptori.
2. Definició del directori de treball.
3. Entrada de les instruccions o pasos de càlcul que hom vulgui executar.

5.2.1 Inicialització de l'intèrpret

Com és ben sabut, per a inicialitzar l'intèrpret només cal fer clic sobre la icona corresponent de l'escriptori. La resposta del computador consisteix en obrir una finestra com la mostrada a la Figura 5.1. El missatge desplegat diu

```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing" Copyright (C)
2013
The R Foundation for Statistical Computing Platform:
i686-pc-linux-gnu (32-bit)

...
```

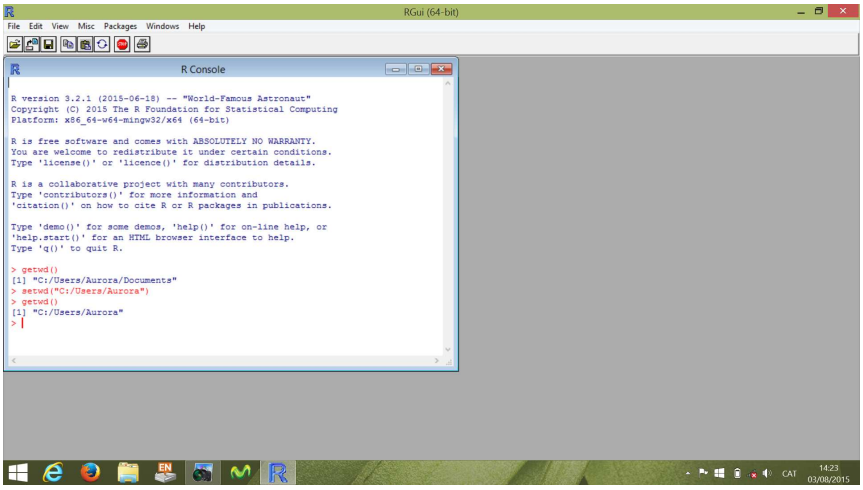

Figura 5.1: Finestra gràfica d'**R** executat en Windows 8.

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Aquest missatge dona diverses informacions sobre el llenguatge **R** que hi ha instal.lat al computador. El símbol > indica que l'intèrpret està esperant l'entrada de comandes via teclat. Aquest símbol s'anomena *prompt*. Notis que, a la pantalla de la Figura 5.1, han estat executades les comandes `getwd()` i `setwd()`. Notis també que la darrera icona a la barra de tasques és la d'**R**.

5.2.2 Definició del directori de treball

En executar l'intèrpret, el sistema operatiu li assigna per defecte un directori de treball, és a dir, un directori on quedaran enregistrades les operacions que es facin. Per tal de no córrer el perill d'efectuar operacions en directoris no desitjats, cal efectuar un control estricte sobre aquest directori. Suposem que avui hom vol treballar en el directori

```
D:/usurais/robert/escola/assignatures/informatica/lab/sessio_1
```

Aleshores es procedirà com segueix. S'identifica el directori de treball, assignat per defecte pel sistema operatiu, amb la comanda

```
> getwd()
```

La resposta serà del tipus

```
[1] "C:/Users/Robert/Documents"
```

Cas que no sigui el directori desitjat, es canvia amb la comanda

```
> setwd("D:/usurais/robert/escola/assignatures
                                    /informatica
                                    /lab/sessio_1")
```

On cal escriure entre cometes (") el nom complet del directori que hom vol que sigui el de treball. Notis que, mentre que la notació estàndar de Windows per designar el nom absolut d'un fitxer empra la contrabarra, \, la notació que cal usar en **R** és la barra normal, /.

5.2.3 Entrada d'instruccions

En lloc de fer cap descripció dificil de redactar i feixuga d'entendre, il.lustrarem aquest punt amb un seguit d'exemples.

Suposem que es vol saber el resultat d'avaluar la fracció $\frac{2}{3}$. Aleshores l'usuari entraria pel teclat la cadena de caràcters `2/3`, que representa la fracció segons la sintaxi d'**R**. La interacció apareixeria en el terminal com

```
> 2/3
[1] 0.6666667
>
```

Un cas un xic més interessant resulta si el que es vol és definir una fracció amb numerador i denominador variables. Aleshores es podria interaccionar amb **R** de la següent manera

```
> x <- 4
> y <- 3
> x/y
[1] 1.333333
>
```

Suposem ara que volem calcular `y/x`. Com que a les variables `x` i `y` ja els hi ha estat assignat un valor, només cal entrar la nova expressió. Tota la interacció resultaria ser

```
> x <- 4
> y <- 3
> x/y
[1] 1.333333
> y/x
[1] 0.75
>
```

En essència, l'intèrpret **R** es comporta com una calculadora de butxaca. Però també es poden definir operacions més complexes. Per exemple

```
> cat("Quin es el teu nom? \n")
"Quin es el teu nom?
> nom <- scan(n=1, what=character(),quiet=TRUE)
1: Robert
> cat("El teu nom es : ", nom, "\n")
El teu nom es :  Robert
>
```

Clarament, aquest darrer exemple no sembla que tingui cap gran utilitat. En efecte, treballar en mode intèrpret resulta especialment útil quan cal fer petites provatures per tal de comprovar característiques del llenguatge **R**, quan es tenen dubtes o bé com a calculadora de butxaca. Per exemple, per veure la diferència de comportament entre els dos operadors de divisió que inclou el llenguatge **R**: la divisió entera, `/`, i la divisió en els reals, `%/%` les comandes serien

```
> x <- 4
> y <- 3
> x%/%y
[1] 1
> x/y
[1] 1.333333
>
```

La sessió de l'intèrpret s'acaba entrant la comanda `quit()` o l'abreujada, `q()`, per exemple

```
> quit()
```

5.3 Treball en mode comanda

En el treball en mode comanda, els programes **R** es processen de manera semblant a com es faria si el llenguatge fos compilat. El procediment consta del quatre passos

1. Inicialització de l'intèrpret **R** a partir de la icona a l'escriptori.
2. Definició del directori de treball.
3. Edició del fitxer que contindrà el codi del programa.
4. Càrrega del fitxer i execució del programa.

Naturalment, els passos 1 i 2 són els mateixos que han estat descrits més amunt. Ara descriurem els passos 3 i 4.

5.3.1 Edició del programa

Editar un programa significa escriure en un fitxer el codi de llenguatge de programació **R** corresponent al programa que resol el problema considerat. L'edició d'un fitxer pot iniciar-se a partir d'un fitxer buit (fitxer inexistent) cas en el qual el fitxer es crearà, o bé a partir d'un fitxer prèviament existent.

En general, editar un fitxer inclou tants processos de modificació del seu contingut com calgui fins arribar a la versió final desitjada.

Encara que no resulta imprescindible, per raons de mnemotècnica, conveniència i possibles compatibilitats entre sistemes operatius diversos, per convenció, els fitxers que contenen programes codificats en llenguatge **R** sempre tindran l'extensió `.R`. El nom `primerPrograma.R` n'és un exemple.

Un cop executat l'intèrpret **R**, per a crear un fitxer nou, hom seguirà un procès com el següent:

1. Esbrinem quin directori de treball ha estat assignat

   ```
   > getwd()
   ```

   Suposem que la resposta sigui

   ```
   C:/Users/Robert/Documents
   ```

   Si no és el que cal, es canvia seguint el procediment descrit a la Secció 5.2.2.

2. Dins l'opció `File` del menú de la part superior de la finestra d'**R** es seleccionarà l'opció `New script`.

3. S'obrirà la finestra de l'editor d'**R**. El nom de la finestra serà alguna cosa semblant a

   ```
   Untitled - R Editor
   ```

4. Ara cal entrar les línies del codi del programa. Per exemple

   ```
   # primerPrograma.R

   cat("Quin es el teu nom? \n")
   nom <- scan(n=1, what=character(),quiet=TRUE)
   cat("El teu nom es : ", nom, "\n")
   ```

5. Ara cal salvar el codi en un fitxer amb nom. A l'opció `File` del menú de la finestra d'**R**, selecciona l'opció `Save as ...`

   S'obrirà una nova finestra en la que s'entrarà el nom del fitxer. Per exemple `primerPrograma.R`, i es validarà. Notis que el títol de la finestra de l'editor ha canviat a

```
C:\Users\Robert\Documents\primerPrograma.R - R Editor
```

que ara fa referència al fitxer que ha estat definit i emmagatzemat.

Després de totes aquestes operacions, el fitxer `primerPrograma.R` ha estat creat i emmagatzemat al directori de treball. Per tal de comprovar-ho, es pot usar la comanda

```
list.files()
```

El resultat de la comanda mostra els noms de tots els fitxers de qualsevol classe que hi hagi al directori de treball. Entre ells hom trobarà `primerPrograma.R`.

Per emmagatzemar noves versions del programa cada cop que es fan més canvis posteriors al primer cop que ha estat salvat, en lloc de l'opció `Save as ...` del menú `File` s'utilitzarà l'opció `Save`.

Quan el que cal fer és editar un programa emmagatzemat en un fitxer previament existent, el procès coincideix amb el que acabamen de descriure llevat del pas 2. Per iniciar l'edició, en compte de seleccionar l'opció `New script` del menú `File` es seleccionarà `Open script ...` i s'obrirà una finestra en la que caldrà introduir el nom del fitxer del programa que cal editar. Anàlogament, les edicions es salvaran usant l'opció `Save`. Si hom vol fer una còpia del fitxer amb un nom diferent, seleccionarà l'opció `Save as ...`

5.3.2 Càrrega i execució

Assumim que mitjançant l'editor de fitxers de l'intèrpret **R**, en el fitxer de nom `primerPrograma.R` ha estat codificat el programa descrit en el pas d'edició. La càrrega del fitxer i l'execució del programa es desencadena amb la comanda

```
> source("primerPrograma.R")
```

Si es desencadena l'execució de l'intèrpret **R** en el directori on està emmagatzemat el fitxer `primerPrograma.R`, l'aspecte de la pantalla resultant seria semblant a

```
> source("primerPrograma.R")
```

```
Quin es el teu nom?
1: Robert
El teu nom es :  Robert
>
```

Suposa ara que la càrrega del programa genera la següent sortida al terminal

```
> source("primerPrograma.R")
Error in source("primerPrograma.R") :
primerPrograma.R:5:8: unexpected symbol
4: nom <- scan(n=1, what=character(),quiet=TRUE)
5: cat(El teu
       ^
>
```

S'ha detectat un error de sintaxi i no s'ha arribat a executar el programa. En efecte. A la primera línia del missatge generat per l'intèrpret s'indica que hi ha un error de sintaxi **R** en el codi que ha escrit el programador (*source code*). L'error ha estat detectat a la línia 5 del codi i, exactament, a la columna 4, indicada pel símbol ^ (accent circumflex). Examinat amb l'editor el codi que contè el fitxer de nom `primerPrograma.R` hom trobaria

```
# primerPrograma.R

cat("Quin es el teu nom? \n")
nom <- scan(n=1, what=character(),quiet=TRUE)
cat(El teu nom es : ", nom, "\n")
```

Clarament a la darrera setència hi manca el símbol doble cometa (") al principi de la cadena de caràcters

```
El teu nom es :  "
```

Amb l'editor s'esmenarà l'errada, el fitxer modificat (editat) es salvarà amb el mateix nom que tenia i es tornarà a carregar i executar. Ara l'execució no tindria problemes i els resultats serien coincidents amb els obtinguts abans.

Suposa ara que, a l'hora de definir el directori de treball s'ha comés un error i s'ha definit un directori diferent d'aquell en el que ha estat executat

l'intèrpret i que, per tant, no conté el fitxer `primerPrograma.R`. El resultat seria alguna cosa semblant a

```
> source("primerPrograma.R")
Error in file(filename, "r", encoding = encoding) :
  cannot open the connection
In addition: Warning message:
In file(filename, "r", encoding = encoding) :
cannot open file 'primerPrograma.R': El fitxer o directori
no existeix
>
```

La línia més rellevant és la darrera la qual indica que el fitxer no existeix. Naturalment, el fitxer no existeix en el directori definit com a directori de treball. Seguint els passos explicats a la Secció 5.2.2 es canvia el directori de treball a aquell que contingui el programa a editar i es torna a intentar la càrrega i execució.

Capítol 6

Treball en Linux

La manera usual de treballar en Linux és via comandes entrades en un terminal de text. Això implica que no es pot usar l'editor de fitxers incorporat en **R** sinó que cal usar un editor de fitxers extern. Si l'usuari de Linux vol treballar via interfase gràfica cal que utilitzi algun dels IDE desenvolupats en el món del llenguatge **R**, per exemple `RStudio`. En el que segueix ignorarem aquest mode de treball. Atès que treballar en un IDE amaga conceptes que cal aprendre per tal que el processament de programes no sembli ser quelcom màgic, considerem que els IDE no són l'eina més adient en un curs introductori.

Suposem que el senyal que dona Linux en un terminal per indicar que està en espera de rebre una comanda és el símbol $. Aquest símbol, com ja ha estat dit, s'anomena amb la paraula anglesa *prompt*. Primer descriurem el treball en mode intèrpret i tot seguit en mode comanda.

6.1 Treball en mode intèrpret

Quan es treballa en sistema operatiu Linux, llenguatge de programació **R** en mode intèrpret, el procediment consta de dos pasos:

1. Inicialització de l'intèrpret **R** per execució de la comanda corresponent del sistema operatiu.

2. Entrada de les instruccions o pasos de càlcul que hom vulgui executar.

El primer pas és ben simple, només cal obrir un terminal i executar la comanda

```
$ R
```

La resposta del computador és alguna cosa semblant a

```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing" Copyright (C)
2013
The R Foundation for Statistical Computing Platform:
i686-pc-linux-gnu (32-bit)

...

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Aquest missatge dona diverses informacions sobre el llenguatge **R**. El símbol $>$ és el prompt de l'intèrpret **R** i, com ha estat dit més amunt pel Linux, indica que l'intèrpret està a l'espera de rebre ordres de càlcul pel teclat. La Figura 6.1 mostra el resultat de l'execució d'**R** en un terminal Linux/Ubuntu.

Pel que fa al segon pas, resulta ser pràcticament el mateix que en el cas de treballar en Windows. Suposem que volem saber el resultat d'avaluar la fracció $\frac{2}{3}$. Aleshores l'usuari entraria pel teclat la cadena de caràcters `2/3` que representa la fracció segons la sintaxi d'**R**. El diàleg apareixeria en el terminal com

```
> 2/3
[1] 0.6666667
>
```

Un cas un xic més interessant resulta si el que volem és definir una fracció amb numerador i denominador variables. Podriem interaccionar amb **R** de la següent manera

```
> x <- 4
```

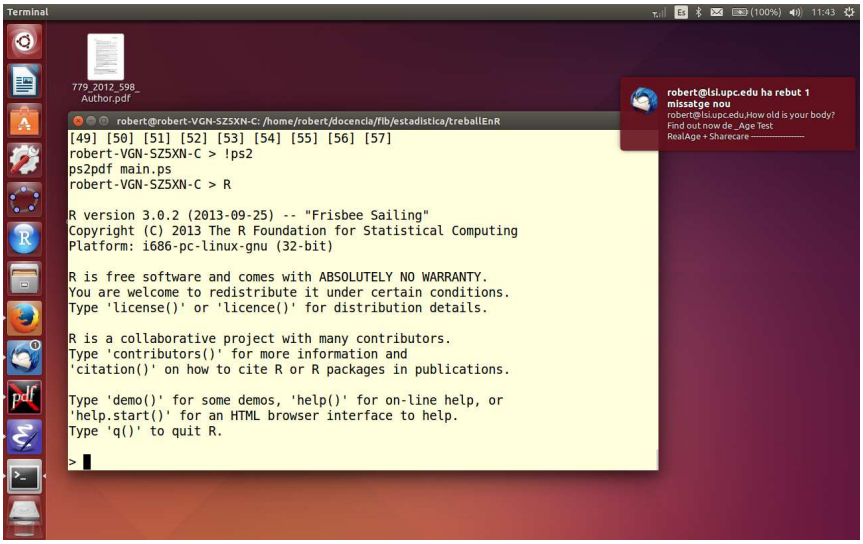

Figura 6.1: **R** executat en un terminal Linux/Ubuntu.

```
> y <- 3
> x/y
[1] 1.333333
>
```

Suposem ara que el que volem es calcular `y/x`. Com que a les variables `x` i `y` ja els hi ha estat assignat un valor, només cal entrar la nova expressió. Resultaria

```
> x <- 4
> y <- 3
> x/y
[1] 1.333333
> y/x
[1] 0.75
>
```

En essència, l'intèrpret **R** es comporta com una calculadora de butxaca. Però també podem definir operacions més complexes. Per exemple

```
> cat("Quin es el teu nom? \n")
Quin es el teu nom?
> nom <- scan(n=1, what=character(),quiet=TRUE)
1: Robert
> cat("El teu nom es : ", nom, "\n")
El teu nom es :  Robert
>
```

Clarament, aquest darrer exemple no sembla que tingui cap gran utilitat. En efecte, treballar en mode intèrpret resulta especialment útil quan cal fer petites provatures per tal de comprovar característiques del llenguatge **R** quan es tenen dubtes o com a calculadora de butxaca. Per exemple, per veure la diferència de comportament entre els dos operadors de divisió que inclou el llenguatge **R**: la divisió entera, `/`, i la divisió en els reals, `%/%`

```
> x <- 4
> y <- 3
> x%/%y
[1] 1
> x/y
[1] 1.333333
>
```

La sessió de l'intèrpret s'acaba entrant una de les dues comandes, `quit()` o abreujada, `q()`, per exemple

```
> quit()
```

6.2 Treball en mode comanda

En el treball en mode comanda, els programes **R** es processen de manera semblant a com es faria si el llenguatge fos compilat.

El punt diferent quan es treballa en Linux és que els programes d'**R** s'editaran usant un editor de fitxers. Hom pot usar qualsevol dels que inclogui la distribució de Linux instal.lada o el que més li agradi dels que es poden

descarregar de la xarxa. Exemple d'editors són `emacs, vi, edit, gedit`. El procediment de treball consta de tres pasos:

1. Definició del programa **R** en un fitxer mitjançant un editor de fitxers. Per convenció, els fitxers que contenen programes codificats en **R** sempre tindran l'extensió `.R`. El nom `primerPrograma.R` n'és un exemple.

2. Execució de l'intèrpret **R** en el terminal del computador mitjançant la comanda

   ```
   $ R
   ```

3. Càrrega del programa en l'intèrpret i execució aplicant la comanda de l'intèrpret

   ```
   > source("nomPrograma.R")
   ```

Suposa que, mitjançant un editor de textos, ha estat codificat dins del fitxer de nom `primerPrograma.R` el següent programa **R**:

```
# primerPrograma.R

cat("Quin es el teu nom? \n")
nom <- scan(n=1, what=character(),quiet=TRUE)
cat("El teu nom es : ", nom, "\n")
```

Si es desencadena l'execució de l'intèrpret **R** en el directori on està emmagatzemat el fitxer `primerPrograma.R`, la interacció mostraria per la pantalla del terminal el següent

```
$ R
...
> source("primerPrograma.R")
Quin es el teu nom?
1: Robert
El teu nom es :  Robert
>
```

Suposa ara que l'execució del programa genera la següent sortida al terminal

```
> source("primerPrograma.R")
Error in source("primerPrograma.R") :
primerPrograma.R:5:8: unexpected symbol
4: nom <- scan(n=1, what=character(),quiet=TRUE)
5: cat(El teu
       ^
>
```

A la primera línia del missatge generat per l'intèrpret s'indica que hi ha un error de sintaxi **R** en el codi (*source*). L'error ha estat detectat a la línia 5 del codi i, exactament, a la columna 4, indicada pel símbol ^ (accent circumflex). Examinat el codi que contè el fitxer de nom `primerPrograma.R` resulta ser

```
# primerPrograma.R

cat("Quin es el teu nom? \n")
nom <- scan(n=1, what=character(),quiet=TRUE)
cat(El teu nom es : ", nom, "\n")
```

Clarament a la darrera setència hi manca el símbol doble cometa (") al principi de la cadena de caràters

```
El teu nom es :  "
```

Un cop esmenada l'errada, una nova càrrega del programa en **R** executarà sense més problemes.

Suposa ara que l'execució de l'intèrpret **R** s'ha efectuat en un directori on no hi ha cap fitxer de nom `primerPrograma.R`. El resultat seria alguna cosa semblant a

```
$ R
...
> source("primerPrograma.R")
Error in file(filename, "r", encoding = encoding) :
  cannot open the connection
In addition: Warning message:
In file(filename, "r", encoding = encoding) :
cannot open file 'primerPrograma.R': El fitxer o directori
no existeix
```

```
>
```

On la línia més rellevant és la darrera la qual indica que el fitxer no existeix. Naturalment, el fitxer no existeix en el directori triat com a directori de treball. El més convenient acostuma a ser tancar la sessió de l'intèrpret, canviar de directori de treball i reiniciar l'intèrpret.

Capítol 7

Metodologia elemental d'anàlisi

Descriurem una metodologia pràctica d'anàlisi d'enuncitas molt senzilla i, per tant, poc potent. Això no obstant, resultarà ser suficientment bona per a tenir èxit en un curs introductori a la programació de computadors en un curriculum on la informàtica és una eina, no l'objectiu.

7.1 Problemes de programació de computadors

En la programació de computadors es distingeixen dues grans families de problemes de programació: la programació de solucions i la resolució de problemes per programació.

En la programació de solucions se suposa que l'enunciat dóna una solució efectiva del problema i tot el que cal fer és expressar-la en forma de programa de computador usant un llenguatge de programació. Per exemple, si el problema és dissenyar un programa que calculi les solucions de polinomis de segon grau

$$ax^2 + bx + c = 0$$

sabent que la solució del problema és

$$x = \frac{-b \overset{+}{-} \sqrt{b^2 - 4ac}}{2a}$$

la programació resulta trivial i la qüestió es redueix a saber que el símbol $\overset{+}{-}$ en general no té representació pròpia en els llenguatges de programació de computadors.

En la resolució de problemes per programació, l'enunciat no dóna la solució al problema. Per programar-la cal primer cercar-ne una. Per exemple, si l'enunciat demana programar el càlcul de la integral

$$\int_a^b x^2 \sin(x) dx$$

i el programador de computadors no sap calcular integrals mitjançant mètodes numèrics, la situació es complica i caldrà primer cercar o inventar la solució matemàtica del problema.

En qualsevol cas, cal coneixer una metodologia d'anàlisi de problemes per a la posterior programació de la solució. A la literatura hi ha diverses tècniques entre les quals s'hi troben l'anàlisi descendent, anàlisi recursiva, per families de problemes, etc. Nosaltres en proposem una de molt senzilla composta de dos pasos: 1) l'observació de les activitats que fem si resolem el problema amb llapis i paper i 2) la posterior sistematització d'aquestes observacions. Considerarem problemes de complexitat baixa o moderada, com correspon a un curs introductori de programació de computadors.

7.2 La metodologia

La metodologia pràctica elemental que proposem té com a hipòtesis bàsiques que

1. L'enunciat és no ambigu, cosa no sempre fàcil d'aconseguir.
2. O bé l'enunciat descriu la solució del problema o bé aquesta solució cau dins dels coneixements que obligatòriament un estudiant del curs que ens ocupa hauria d'haver adquirit en la seva formació prèvia a haver arribat al curs actual.

La metodologia consta dels següents pasos:

1. Llegeix l'enunciat acuradament, sense pressa i tans cops com calgui per asseguar-te'n de que saps identificar quin problema cal resoldre, quines dades dona l'enunciat i quins son els resultats exigits. Mai menystinguis aquest apartat per senzill que et sembli.
2. Identifica exactament les dades que donen. Per a cadascuna de les dades que representi una informació variable, assigna-li un nom concret

amb un mnemònic que la identifiqui. Usa noms compostos ùnicament per lletres, digíts àrabs i guionet baix. Per exemple

`mitjana`	:	Mitjana d'una població
`variancia_escolar`	:	Variància escolar
`variancia_professional`	:	Variància professional

No cal assignar noms massa llargs, però tampoc et preocupis gaire, un cop picats la primera vegada, després es fàcil replicar-los.

De vegades resulta molt útil assignar noms de variable a informacions que de fet són constants. En aquest cas una bona pràctica és assignar-los-hi noms escrits amb majúscules per diferenciar-los de les variables verdaderes. Per exemple

`G`	:	Gravetat terrestre
`KCG`	:	Constant compensadora de Galbraith
`FS`	:	Factor Samuelson

3. Identifica exactament els resultats exigits. Obviament, com que cal calcular-lo, un resultat no pot ser una informació constant i caldrà representar-lo per una variable. A cada component d'un resultat cal assignar-li un nom concret amb un mnemònic que la identifiqui.

4. Pensa com resoldries el problema si ho fessis amb llapis i paper de manera metòdica emprant els noms de variables i constants que has identificat als passos prèvis. Escriu sobre un paper el procediment que has pensat seguint un format de recepta de cuina però amb la màxima precisió possible, és a dir, usant una escriptura tan semblant com puguis a la sintaxi del teu llenguatge de programació.

5. Reflexiona sobre el teu procediment de càlcul. Si cal modifica'l tants cops com calgui fins que n'estiguis convençut de que descriu la solució del problema plantejat.

6. Prepara un banc de prova per a la teva solució. Això implica preparar les dades per un mínim de tres o quatre casos diferents pels quals o bé coneixes la solució o bé ets capaç de calcular-la manualment.

7. Codifica en el computador el teu procediment segons la sintaxi i la semàntica del llenguatge **R**. No és aconsellable utilitzar expressions innecessàriament complicades. Recomanem usar variables auxiliars que representin càlculs intermedis. Per tal d'il.lustrar aquesta idea, considera el següent enunciat:

   > *En un sistema de control d'alarmes hom diu que quan hi ha una alarma aquesta està* activada. *El sistema es regeix per les següents regles:*
   >
   > *(a) Cada alarma pot estar activada o no independentment de les altres dues.*
   >
   > *(b) Es considera que hi ha alarma real quan una i només que una de les tres alarmes està activada.*
   >
   > *Dissenya un programa de computador tal que rep com a dades el valor lògic de cadascuna de les tres alarmes individuals i decideix si hi ha o no alarma real.*

   Si definim les variables

`a1`	:	Primera alarma
`a2`	:	Segona alarma
`a3`	:	Tercera alarma

   Una possible solució al problema seria

```
 1 # Control d'alarmes
 2
 3 a1 <- scan(n=1, what=logical(), quiet=TRUE)
 4 a2 <- scan(n=1, what=logical(), quiet=TRUE)
 5 a3 <- scan(n=1, what=logical(), quiet=TRUE)
 6
 7 if (( a1 && !a2 && !a3) ||
 8     (!a1 &&  a2 && !a3) ||
 9     (!a1 && !a2 &&  a3)){
10    cat("Hi ha alarma \n")
11 }else{
12    cat("No hi ha alarma \n")
13 }
```

La condició lògica que governa la sentència `if` a la línia 7 és compacta perquè utilitza el nombre mínim de variables i de línies de codi. Alhora, però, és complexa i per tant costosa de pensar i d'escriure. El principal problema però sorgeix quan el programa genera resultats incorrectes i cal raonar sobre la seva correctesa. Això implica haver de decodificar el codi previament escrit, és a dir, tornar a entendre que és el que haviem codificat.

Atès que en un curs introductori l'objectiu no és escriure programes òptims ni en espai ocupat ni en temps de càlcul, aconsellem descomposar càlculs complexos en diversos passos més senzills. Per a l'exemple precedent, afegiriem les següents variables

`a_nomes1`	:	Activada només la primera alarma
`a_nomes2`	:	Activada només la segona alarma
`a_nomes3`	:	Activada només la tercera alarma
`a_real`	:	Existeix alarma real

Ara la solució seria

```
 1 # Control d'alarmes
 2
 3 a1 <- scan(n=1, what=logical(), quiet=TRUE)
 4 a2 <- scan(n=1, what=logical(), quiet=TRUE)
 5 a3 <- scan(n=1, what=logical(), quiet=TRUE)
 6
 7 a_nomes1 <-  a1 && !a2 && !a3
 8 a_nomes2 <- !a1 &&  a2 && !a3
 9 a_nomes3 <- !a1 && !a2 &&  a3
10 a_real   <- a_nomes1 || a_nomes2 || a_nomes3
11
12 if(a_real){
13    cat("Hi ha alarma real \n")
14 }else{
15    cat("No hi ha alarma real \n")
16 }
```

Si cal comprovar la validesa dels càlculs, quina de les dues codificacions podràs analitzar més fàcilment? Recorda que

> *Durant el desenvolupament, un programa de computador es llegeix moltíssimes més vegades que no s'executa.*

8. Executa el teu programa i depura totes les errades de sintaxi que apareguin. Aquest pas és rutinari.

9. Executa el teu programa i depura totes les errades d'execució que apareguin, per exemple, un intent de divisió per zero o el càlcul del logaritme d'un valor negatiu. Aquest pas és rutinari.

10. Executa el teu programa sense errors sintàctics ni d'execució. Ara ja s'obtenen resultats. Mentre els resultats no siguin els esperats, el teu programa és erròni. Aquesta és la situació més complicada en el desenvolupament d'un programa de computador i és, amb diferència, el punt on més esforç caldrà esmerçar. S'afrontarà de manera tan sistemàtica com sigui possible.

    (a) Comprova que has codificat el que volies pel que fa a les estructures de control: seqüències, condicionals i iteracions.

    (b) Comprova la sintaxi de les expressions de càlcul, especialment aquelles que inclouen parèntesis.

    (c) Inclou sentències de sortida (`cat()`) al teu codi per poder analitzar com evolucionen els valors de les teves variables i aïllar on es produeixin càlculs incorrectes. Aquesta tècnica s'anomena *fer la traça* del programa perquè traça l'evolució dels valors de les variables.

    (d) Per a cada punt on els resultats parcials siguin incorrectes, aplicarem tot el procés d'anàlisi que hem descrit, arribant si cal, a haver de reinterpretar l'enunciat (pot ser no l'haviem entés bé) i a redefinir variables i càlculs intermedis.

11. Ara els resultats pel banc de proves són correctes. Vol dir això que el programa és correcte? La resposta és, en general, **negativa**. El conegut principi de la Matemàtica en el qual es basa la tècnica de demostració per reducció a l'absurd té ací un enunciat paradigmàtic:

    > *Un nombre indefinit de casos de prova pels quals un programa de computador genera resultats correctes no demostra que el programa ho sigui. Un sol cas en el qual el programa genera resultats incorrectes demostra que el programa és incorrecte.*

Una consideració final. En la metodologia emprada en el món de la simulació per construir models de sistemes, el procès mitjançant el qual s'aconsegueix que el model generi resultats correctes per a dades conegudes s'anomena *validació*. Per intentar garantir-ne la correctesa, un cop un model ha estat validat, i sempre que es pugui fer, el model s'aplica a situacions amb dades per a les quals no es coneix la solució. Tot seguit es va al laboratori i es desenvolupa un assaig amb el sistema real i se li apliquen les dades amb solució desconeguda. Si els resultats de l'assaig coincideixen amb els produïts pel model, hom diu que el model ha estat *verificat* i es dóna com a correcte.

A la programació de computadors es pot aplicar el mateix principi. Un cop un programa genera resultats correctes per tots el casos del banc de proves, el programa s'aplica a unes dades per a les quals no es coneix la solució. Ara el problema es resol manualment o per altres mètodes, sempre que sigui possible, per a les dades de resultat desconegut. Finalment es comparen els resultats generats pel programa amb els obtinguts manualment. Si els resultats són coincidents hom diu que el programa es *raonablement* correcte.

Com? Encara no podem dir que el programa és correcte? Efectivament, en general no, no ho podem dir. El principi de contradicció continua aplicant-se-hi. En fi, deixem la discusió d'aquesta qüestió pels matemàtics i els teòrics de la informàtica. Però tinguem-ho sempre present en els estudis i en la vida de cada dia: que una *" cosa"* estigui *"feta amb computador"* no implica que necessàriament sigui correcta!

Epíleg

Careful design is great. Testing is great. Formal methods are great. Code reviews are great. Static analysis is great. But none of these things alone are sufficient to eliminate bugs[1] *: They will always be with us. A bug can exist for half a century despite our best efforts to exterminate it.*

Joshua Bloch, *Extra, Extra—Read All About It: Nearly All Binary Searches and Mergesorts are Broken*, Posted: Friday, June 02, 2006.

[1] A bug is a flaw or fault in a program that produces unexpected results.

Apèndix A

Algunes comandes útils de l'entorn R

L'intèrpret **R** inclou tot un conjunt de comandes de consola que permeten navegar per l'entorn **R** i manegar l'espai de treball, és a dir, un conjunt de fitxers presents, en principi, en el directori de treball. Tot seguit donem una taula d'aquelles comandes que resulten més útils en un curs d'iniciació a la programació en llenguatge **R**.

dir() Llista els fitxers del directori de treball.

dir.create(nom)
Crea un fitxer de nom absolut definit per `nom`. Per exemple `" C:/Usuaris/Robert/prova"`.

getwd() Retorna el nom absolut del directori de treball.

history() Llista les darreres 25 comandes entrades a la consola.

list.files() Llista els fitxers del directori de treball.

load("nomFitxer.RData")
Carrega l'espai de treball emmagatzemat a `nomFitxer.RData`.

loadhistory("nomFitxer.Rhistory")
Carrega la història de comandes del fitxer `nomFitxer.Rhistory`.

ls() Llista les variables definides en l'espai de treball.

quit() Acaba la sessió d'**R**.

q() Acaba la sessió d'**R**.

rm(x) Elimina la variable x de l'espai e treball.

rm(list=ls()) Elimina tota la informació de l'espai de treball.

savehistory("nomFitxer.Rhistory")
Salva la història de comandes en `nomFitxer.Rhistory`.

save.image() Salva l'espai de treball en el fitxer amb extensió `.RData` que hi hagi al directori de treball.

save.image("nomFitxer.RData")
Salva l'espai de treball específicament en el fitxer `nomFitxer.RData` del directori de treball.

setwd(nomAbsolut)
Defineix com a directori de treball el donat per `nomAbsolut`. Per exemple `setwd(" C:/docs/noudir")` en Windows i `setwd("/usr/rob/noudir")` en Linux.

setwd("..") Defineix com a nou directori de treball el pare de l'actual directori de treball.

setwd("nomRelatiuWD")
Defineix com a nou directori de treball el directori que té com a nom relatiu `nomRelatiuWD` respecte del directori de treball actual.

source("nomFitxer.R")
Carrega en l'intèrpret i executa el programa del fitxer `nomFitxer.R`.

Tens una necessitat que no resol cap de les comandes de la llista? Aleshores:

1. Pensa unes poques paraules clau que defineixin la teva necessitat. Inclou sempre, si més no, aquestes tres: `R, language, environment`
2. Obre el teu navegador i demana-li que faci una cerca amb les paraules claus pensades.

3. Entre el munt de cites que trobarà el navegador, selecciona aquelles que puguin respondre al teu problema.

4. Si no tens èxit, analitza les paraules que hi ha a les cites trobades pel navegador i usa-les per a afinar les teves claus de cerca. Ara torna a cercar.

Bibliografia

1. J. Baixeries, N. Pallarès, S. Pérez, E. Romero. *Exercicis. Introducció a la Informàtica. Grau d'Estadística*. Notes del curs.

2. Michael J. Crawley. *The R Book*, 2nd edition, A John Wiley & Sons, Ltd., 2013. Hom pot descarregar el llibre, entre d'altres, de la seu web `http://serverlib.moe.gov.ir/documents/10157/42675/The+R+Book+.pdf`.

3. N. Pallarès, J. Baixeries. *Manual de laboratori. Introducció a la Informàtica. Grau d'Estadística*. Notes del curs.

4. Roger D. Peng. *R Programming for Data Science*. Hom pot descarregar el llibre de la seu web `http://leanpub.com/rprogramming`.

5. *The Comprehensive R Archive Network*. `https://cran.r-project.org/`. Disponible el juliol 2015.

6. *The R Project for Statistical Computing*. `https://www.r-project.org/`. Disponible el juliol 2015.

7. *RStudio*. `https://www.rstudio.com/`. Disponible el juliol 2015.