TEMES CLAU

Marta Gatius
Josefina López
Àngela Martín
Óscar Romero
Alfons Valverde
Fatos Xhafa
Ignasi Esquerra
Gerard Amirian

PROGRAMACIÓ PRÀCTICA EN C++

TEMES CLAU 16

PROGRAMACIÓ PRÀCTICA EN C++

Marta Gatius
Josefina López
Àngela Martín
Óscar Romero
Alfons Valverde
Fatos Xhafa
Ignasi Esquerra
Gerard Amirian

Aquesta obra compta amb el suport de la Generalitat de Catalunya

En col·laboració amb el Servei de Llengües i Terminologia de la UPC

Disseny de la coberta: Ernest Castelltort
Disseny de col·lecció: Tono Cristòfol
Maquetació: Mercè Aicart

Primera edició: febrer de 2010

Edicions de la Universitat Politècnica de Catalunya, SL
Jordi Girona Salgado 1-3, 08034 Barcelona
Tel.: 934 137 540 Fax: 934 137 541
Edicions Virtuals: www.edicionsupc.es
E-mail: edicions-upc@upc.edu

Dipòsit legal: B-9575-2010
ISBN: 978-84-9880-403-4

Índex

5 Taules

Àngela Martín

6 Estructures

Oscar Romero

Ignasi Esquerra, Gerard Amirian i Alfons Valverde

Introducció

L'objectiu d'aquest llibre és facilitar la introducció a la programació. Amb aquest llibre pretenem no només fer més fàcil l'aprenentatge dels conceptes fonamentals de la programació i d'un dels llenguatges amb més projecció (el C++), alhora volem potenciar les capacitats intel·lectuals generals necessàries per la programació, especialment l'abstracció i el raonament lògic. També volem satisfer l'interès de l'alumne per aprendre informàtica i donar-li experiència en l'ús de la programació. Amb la finalitat d'aconseguir aquests objectius, s'ha incorporat a cada capítol del llibre nombrosos exemples de programes resolts en els que s'explica de forma detallada l'aplicació dels conceptes bàsics presentats.

El llibre està orientat als cursos d'introducció a la informàtica dels estudis d'enginyeries no informàtiques en general, si bé es pot utilitzar en estudis d'altres nivells, com últims cursos de batxillerat. Ha estat escrit per vuit autors amb una amplia experiència en l'ensenyament de la programació, així com en altres àmbits relacionats amb el desenvolupament de software. El llibre recull, doncs, les experiències dels autors adquirides durant molts anys de classes i d'altres activitats professionals.

En la elaboració del llibre s'han considerat les noves metodologies d'ensenyament proposades en el marc de l'espai europeu d'educació superior (EEES)(avaluació continuada de l'aprenentatge, incentivar la participació de l'estudiant a classe,adaptació als diferents nivells de coneixements i d'interès de l'alumnat...) i les tecnologies que ja s'estan utilitzant a la nostre universitat per facilitar la seva exploració, concretament una plataforma interactiva (atenea). Es per això que el llibre incorpora no solament programes resolts, sinó també execicis proposats per assimilar els nou conceptes, i ajudes per resoldre'ls.

El llibre consisteix en set capítols. En el primer capítol, després d'una breu introducció a la informàtica en general s'expliquen els conceptes de llenguatge de programació, sistemes de numeració i àlgebra de Boole. Als cinc capítols següents es presenten els conceptes fonamentals per a realitzar programes: operadors i tipus de dades, seqüències, accions i funcions, taules i estructures. Aquests cinc capítols segueixen la mateixa estructura: presentació dels nous conceptes, programes en els que s'explica de forma detallada com s'apliquen aquests conceptes, exercicis dirigits en els que es guia a l'estudiant amb indicacions o solucions incompletes, conceptes i problemes avançats i finalment, una llista d'exercicis proposats. L'últim capítol del llibre inclou exemples d'aplicacions amb fitxers àudio, amb fitxers d'imatges i amb targes d'entrada i sortida.

Alguns dels exemples utilitzats al llarg del llibre són programes clàssics (apareixen en altres llibres), altres corresponen a exercicis utilitzats per avaluar el seguiment de l'assignatura en els cursos d'introducció a la informàtica dels estudis d'enginyeria industrial i aeronàutica de Terrassa.

Amb la finalitat de fer més amena la lectura del llibre, certa informació complementària s'ha marcat amb el quadradet de color, que indica que el fragment de text en qüestió es pot llegir de forma independent de la resta del text (inclús es podria obviar).

1 Conceptes bàsics

1.1 Què és la informàtica

El terme *informàtica* és un acrònim de les paraules *informació* i *automàtic* i, per tant, es pot entendre com la ciència que estudia el tractament automàtic de la informació. De tota manera, darrere d'aquesta definició aparentment senzilla, es troben un gran conjunt d'avenços científics i tecnològics que l'home ha i va desenvolupant constantment al llarg de la història, sense oblidar que l'objecte central de tot això és l'ordinador.

La paraula *informàtica* és originària de la francesa *informatique*, inventada per l'enginyer Philippe Dreyfus el 1962.

> La informàtica es pot definir com la disciplina tecnocientífica encarregada de l'estudi de l'ordinador com a mitjà pel processament de la informació.

Els àmbits d'estudi de la informàtica són molt diversos: disseny de bases de dades, càlcul científic i estadístic, comunicacions, arquitectura d'ordinadors, etc. En aquest llibre, ens centrarem en la programació dels ordinadors, en què assentarem les bases per al disseny de petites aplicacions.

1.2 Perspectiva històrica de l'ordinador

Malgrat que l'ordinador com a tal és un invent tecnològic modern, es considera que l'ENIAC és el primer ordinador, el qual es va construir al 1945, tot i que va haver-hi un desenvolupament previ fins arribar en aquest punt. Així, si bé l'ENIAC va marcar l'inici de la història en l'evolució dels ordinadors, que normalment es classifica per generacions, hi havia uns orígens o antecedents previs.

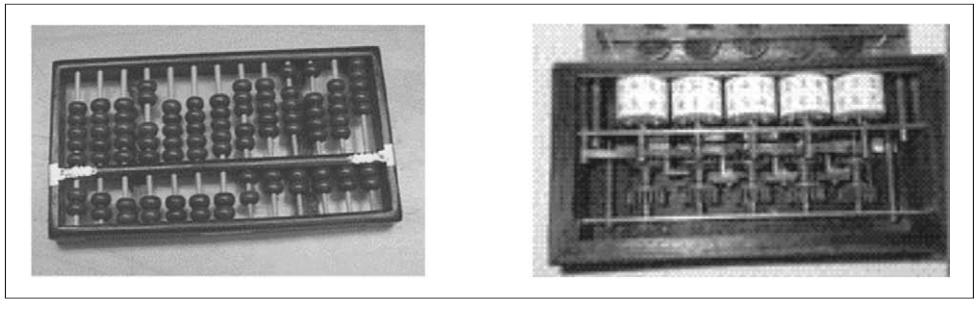


Fig. 1.1 Àbac (esquerra). Màquina aritmètica de Pascal (dreta)

L'origen de l'àbac és incert, malgrat que es creu que és originari de la Xina i hi ha constància del seu ús cap al 400 aC.

Antecedents: Cal destacar els primers instruments de càlcul, com l'*àbac*, que se situa sobre l'any i que és un instrument mecànic per a representar els números i fer determinades operacions aritmètiques bàsiques. Molt després, a partir del segle XVII es comencen a desenvolupar màquines calculadores mecàniques, basades en rodes dentades que funcionen sobre eixos i mecanismes rotatoris. Així, per exemple, el 1642 Pascal va construir la seva *màquina aritmètica*, capaç de fer sumes i restes, i el 1694 Leibnitz va construir la *màquina universal*, que feia les quatre operacions matemàtiques bàsiques.

La màquina creada per Pascal fou anomenada *pascalina*.

Entre 1832 i 1835, Charles Charles Babbage va dissenyar la seva *màquina analítica*, que era com un ordinador mecànic ja que n'incorporava els components bàsics: memòria, unitat de càlcul, unitat de control i unitats d'entrada i sortida, capaç de fer operacions mitjançant targetes perforades que feien de "programes", per la qual cosa va assentar les bases de l'ordinador actual.

Charles Babbage no va construir la seva màquina analítica i va ser Lady Ada Augusta Byron (1815-1852) qui en va recuperar els dissenys; és per això que se la considera la primera programadora de la història i el llenguatge de programació Ada (1980) va rebre aquest nom en honor seu.

El 1889, Hans Hollerith va fer el cens dels Estats Units mitjançant una màquina amb un sistema electrònic per a la lectura de les targetes perforades i un sistema mecànic per a fer els càlculs. El 1944, Howard Aiken amb IBM va construir el MARK 1, la primera màquina electromecànica capaç de resoldre equacions lineals i d'operar amb números de fins a 23 dígits en segons.

Primera generació (1940-1952): Comença el desenvolupament de l'electrònica amb les vàlvules de buit i l'any 1945 John Eckert i John Mauchly presenten a la Universitat de Pensilvània *ENIAC*, que es considera la primera computadora electrònica de propòsit general i que es programava manualment i en llenguatge màquina.

ENIAC és l'acrònim d'*electronic numerical integrator and calculator*.

L'ENIAC pesava 30 tones, tenia 18.000 vàlvules i ocupava 150 m^2.

Més tard, va millorar el projecte Johannes von Neumann, que va fer el primer disseny amb l'emmmagatzematge de les dades i el programa a la memòria.

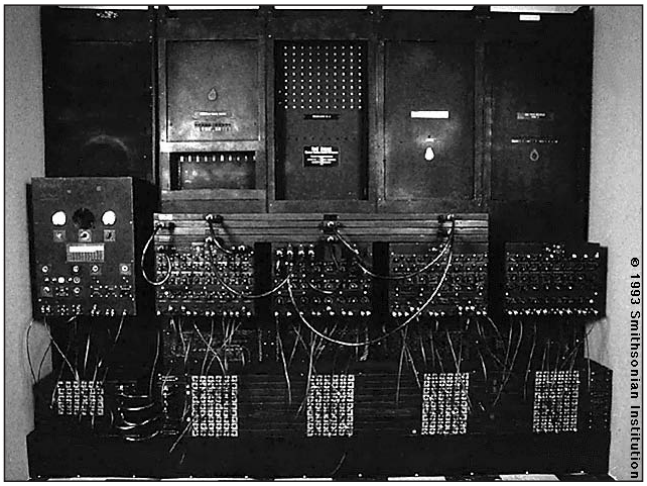


Fig. 1.2 ENIAC

Segona generació (1952-1964): Es produeix la substitució de la vàlvula de buit pel transistor i es passa de mesurar la velocitat en mil·lisegons a microsegons. Apareixen els llenguatges *assembladors* i els llenguatges de programació evolucionats (COBOL, ALGOL, FORTRAN).

> Els llenguatges assembladors són mnemotècnics de les instruccions del llenguatge màquina.

Es desenvolupen instruments d'emmagatzemament d'informació en forma magnètica: els nuclis de ferrita com a memòria interna, i el tambor magnètic i la cinta magnètica com a memòria massiva. Apareixen les primeres empreses a crear ordinadors transistoritzats com IBM i NCR.

> L'any 1943, Thomas Watson, president d'IBM, va dir que "al món com a molt hi ha mercat per a quatre o cinc ordinadors".

Tercera generació (1964-1971): Apareix el circuit integrat i la miniaturització consegüent. La velocitat es mesura en nanosegons. Es desenvolupen els sistemes operatius, les memòries semiconductores i els discs magnètics. Apareixen nous lleguatges de programació, com el BASIC i el PASCAL, i es desenvolupen nous conceptes i tècniques de programació, com la multiprogramació i el temps compartit. El 1970 es presenta l'IBM 370.

Quarta generació (1971-1981): Apareix el microprocessador, la qual cosa afavoreix la popularització de l'ordinador i arribar al concepte d'ordinador personal.

> El 1972, Intel va introduir el concepte de microprocessador.

Apareix el disquet, i comença el desenvolupament de les xarxes de dades i la interconnexió de computadors.

> El disquet o *floppy disk* era un disc magnètic flexible que permetia traslladar dades entre ordinadors.

El 1977, apareix APPLE I, el 1980 la marca Sinclair crea el ZX 80, i el 1983 apareix el Commodore 64. El 1976 apareix el CRAY 1, un dels primers superordinadors.

> El 1981, es va presentar el primer PC d'IBM amb un microprocessador d'Intel i un sistema operatiu de Microsoft, anomenat MS-DOS.

Cinquena generació: Comença el desenvolupament de la informàtica tal com es coneix ara, amb llenguatges orientats a objectes: C++ (1986) i Java (1995), el desenvolupament d'Internet, les aplicacions multimèdia, ordinadors portàtils i el desenvolupament dels microprocessadors i els perifèrics.

> El 1985, Microsoft va presentar la primera versió de Windows, però no va ser fins el 1990 que amb la versió 3.0 va aconseguir implantar-se al mercat.

1.3 Arquitectura d'un ordinador

Des del punt de vista d'un futur programador, és aconsellable disposar d'una mínima base teòrica prèvia sobre l'estructura, el funcionament i l'organització d'un ordinador, que és el que normalment s'estudia en parlar de l'arquitectura d'un computador.

Un ordinador és una màquina electrònica amb la capacitat de processar informació de manera automàtica. El conjunt de dispositius que donen entitat física a la màquina constitueix la part *hardware* o física i tots ells es poden classificar dintre de tres tipus, tal com es representa a la figura 1.3.

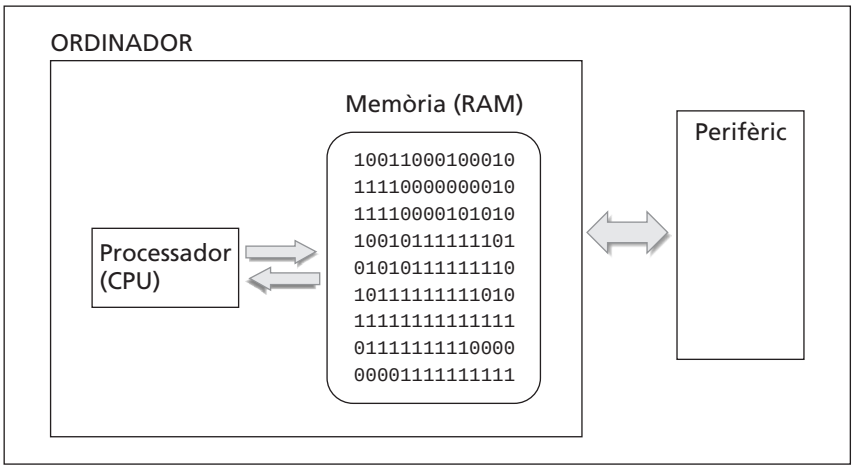


Fig. 1.3 Elements d'un ordinador

Processador (CPU): És el cervell de l'ordinador i, per tant, s'encarrega d'interpretar i executar les ordres rebudes (instruccions) i de dur a terme el processament de les dades.

La Llei de Moore afirma que cada dos anys es dupliquen la potència i la velocitat dels microprocessadors.

Memòria (RAM): És el lloc on l'ordinador emmagatzema les instruccions i les dades que està utilitzant a l'hora de dur a terme una tasca concreta.

Dispositius d'entrada i sortida (perifèrics): Són tots aquells elements que permeten a l'ordinador comunicar-se amb el seu entorn (teclat, pantalla, disc dur, targeta de xarxa, etc).

Un ordinador té dues parts: la part hardware i la part software.

Es pot observar que s'ha emprat el terme *instruccions* per fer referència a aquelles ordres que l'ordinador ha d'executar per dur a terme un procés o tasca. Precisament, un conjunt ordenat i ben definit d'instruccions, necessàries perquè un ordinador desenvolupi una tasca determinada, és el que s'anomena *programa*, que constitueix la part *software* o "tova" d'un ordinador.

> Un ordinador és una màquina electrònica digital capaç de realitzar tasques complexes a gran velocitat, a partir d'uns càlculs numèrics elementals.

Per tant, un ordinador és un conjunt de dispositius físics de naturalesa electrònica que necessita programes per poder funcionar. Aquests programes són específics per a la tasca que l'ordinador hagi d'executar i atès que aquest es basa en elements electrònics de naturalesa digital, els programes són una successió ordenada d'instruccions codificades en llenguatge binari (zeros i uns), que és el que es coneix com a *llenguatge màquina*.

Per a un ordinador, l'ordre de fer una suma en llenguatge màquina pot ser: 11001101.

> Un programa és una successió d'ordres donades a l'ordinador mitjançant les quals se li especifica el que ha de fer amb la informació i com ho ha de fer.

D'altra banda, és important assenyalar que quan un ordinador executa un programa determinat i, per tant, duu a terme tot el que aquest li diu, aquest programa i les seves dades associades han d'estar sempre a la memòria, ja que és a partir d'aquesta que el processador comença a treballar. Es diu llavors que s'ha "carregat el programa a la memòria" quan un programa es col·loca a la memòria perquè el processador hi pugui treballar. Anàlogament, es diu que s'ha "desat un programa" quan des de la memòria es desa el programa en un dispositiu d'emmagatzematge massiu (normalment, el disc dur), ja que la memòria és un dispositiu d'emmagatzematge de tipus volàtil, és a dir, la informació que conté es perd quan aquesta no està alimentada amb el subministrament elèctric necessari.

Normalment, es fa la comparació entre un ordinador i un oficinista, de manera que el processador és l'oficinista en si mateix, la memòria és la seva taula de treball i els dispositius d'entrada i sortida són les finestres per rebre els clients, les llibreries per dipositar els fitxers i la resta d'elements propis d'una oficina. L'oficinista treballa de manera que no pot desenvolupar cap tasca que no quedi especificada concretament a la seva taula de treball mitjançant un llibre o dossier (són els programes de l'ordinador). Així, quant més gran sigui la seva taula de treball, més eficaçment i millor treballarà aquest oficinista.

1.4 Llenguatge de programació

Ja s'ha vist que un ordinador pot desenvolupar tasques gràcies als programes. El problema ara és determinar com es poden construir aquests programes, que és la feina pròpia del programador. Prèviament, s'ha de partir de la base que l'ordinador treballa en llenguatge binari (zeros i uns) i, per tant, qualsevol cosa que se li hagi de comunicar ha de ser en aquest llenguatge.

> El llenguatge màquina és el propi de la màquina i està constituït només per dos símbols: 1 i 0, és a dir, és un llenguatge binari.

Efectivament, fa molt de temps, els programadors feien els seus programes a base de zeros i uns, però això era una feina poc pràctica i ineficaç. Ràpidament, es va pensar que fos el propi ordinador el que fes la tasca de traducció d'un programa al seu llenguatge natural, el llenguatge binari. D'aquesta manera, es podrien confeccionar els programes en un llenguatge similar al humà (sempre prenent com a referència l'anglès), molt més fàcil per a l'ésser humà. Aquest llenguatge hauria de tenir el seu propi vocabulari (òbviament, molt reduït) i la seva pròpia sintaxi. Van començar a aparèixer diferents llenguatges, cadascun d'ells amb els seus avantatges i inconvenients, i tots ells dintre del que es coneix com a "llenguatges d'alt nivell", pel fet que es basen en el llenguatge humà, a diferència del "llenguatge màquina" o de baix nivell, propi de l'ordinador.

> En un llenguatge d'alt nivell, l'ordre suma pot ser l'operador de la suma: $+$.
>
> El llenguatge assemblador (*assembler*) es considera de nivell intermedi. En aquest llenguatge, l'ordre suma pot ser: ADD.

Per tant, a l'hora de confeccionar un programa, es fa normalment amb un llenguatge d'alt nivell i el propi ordinador, mitjançant un altre programa anomenat *compilador*, s'encarrega de traduir-lo al seu propi llenguatge (vegeu la figura 1.4):

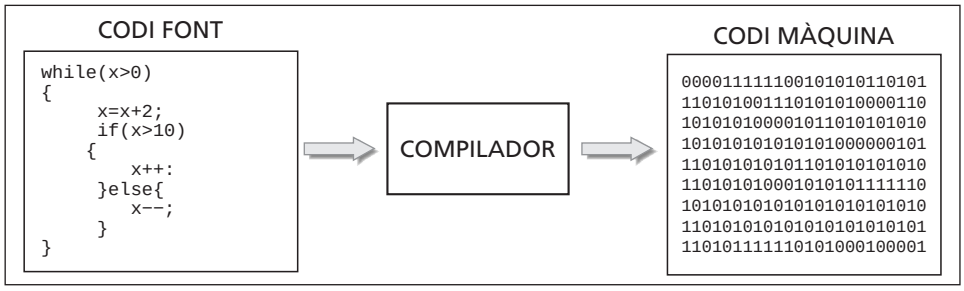


Fig. 1.4 Compilació d'un programa

El compilador és el programa que permet traduir un programa de llenguatge d'alt nivell (fet per nosaltres) a llenguatge màquina (executable per l'ordinador).

1.4.1 Llenguatges de programació

Hi ha diferents tipus de llenguatges de programació d'alt nivell, els quals es poden classificar segons diferents criteris. Des del punt de vista de la seva estructura constitutiva, es pot fer la classificació següent:

Llenguatges imperatius: Utilitzen la instrucció o ordre com a unitat bàsica de treball per indicar a la màquina el que ha de fer. Destaquen els llenguatges: FORTRAN, COBOL, PASCAL, C, ADA, BASIC, LOGO, etc.

Una instrucció també es coneix com a sentència.

Llenguatges declaratius: Utilitzen descripcions o expressions lògiques per indicar a la màquina l'objectiu. Es basen en regles i fets. Dintre d'aquest tipus, destaquen el LISP, el PROLOG i el SQL.

Llenguatges orientats a objectes: Són els que utilitzen "entitats" i les seves funcionalitats per indicar a la màquina el que ha de fer. Destaquen els llenguatges: C++, JAVA i PYTHON.

Llenguatges orientats al problema: Són els que utilitzen les aplicacions de gestió com ara el SQL.

Llenguatges naturals: Són els que intenten resoldre els problemes des del punt de vista del llenguatge humà i s'apliquen en temes com la intel·ligència artificial. Destaquen el LISP i el PROLOG.

FORTRAN és un acrònim de *formula translator*.
COBOL és l'acrònim de *common business oriented language*.
BASIC és l'acrònim de *beginner's all purpose symbolic instruction code*.
PASCAL prové de Blaise Pascal, pensador del segle XVII.

Històricament, el FORTRAN, que va aparèixer el 1955, es pot considerar el primer llenguatge d'alt nivell dirigit a fer aplicacions tècniques i científiques. Posteriorment, el COBOL, apareix el 1960 i és el llenguatge més utilitzat en aplicacions de gestió. El BASIC, neix el 1965 com el primer llenguatge de tot ús. Va ser amb el PASCAL l'any 1970 que es van assentar les bases de la programació estructurada, i posteriorment van sortir altres llenguatges com MODULA-2, ADA i C, que el 1972 va aconseguir la portabilitat de codi, com també la independència del sistema operatiu (plataforma) utilitzat. A partir de llavors, van sortir altres llenguatges com el C++ i el JAVA, orientat a objectes, i el PROLOG i i el LISP, orientats a la intel·ligència artificial (IA).

El pare de la IA és Alan Turing (1921-1954), creador del "test de Turing", que determina el grau d'intel·ligència d'una màquina.

1.4.2 Llenguatge C++

El llenguatge C es va desenvolupar als laboratoris Bell d'AT&T per Dennis Ritchie el 1972, a partir del llenguatge B fet per Ken Thompson el 1970. És un llenguatge que permet programació estructurada amb diversitat d'operadors i tipus de dades, és molt transportable i de gran eficàcia. Posteriorment, el 1985, després d'un redisseny del C, apareix el llenguatge C++, evolució del C que incorpora nous conceptes i característiques, com les *classes* i la programació orientada als objectes.

La programació estructurada només fa servir tres tipus d'instruccions: seqüències, iteracions i seleccions.

El llenguatge C++ és un llenguatge compilat, és a dir, un cop s'ha escrit el codi corresponent a un programa (fase d'edició), mitjançant un programa compilador l'ordinador fa la traducció a llenguatge màquina del programa fet (programa en codi font), de tal manera que se n'obté un altre programa equivalent escrit en binari (programa en codi màquina). A més d'aquesta traducció, s'hi han d'incorporar aquelles informacions que es fan servir en un programa i que no s'han hagut de fer perquè ja estan fetes i s'han incorporat directament al programa (instruccions típiques d'entrada i sortida, de tractament matemàtic, etc.), que globalment s'anomenen *biblioteques de funcions*.

Es diu també *llibreria* en comptes de *biblioteca de funcions*, com a traducció literal de l'anglès *library*.

Aquesta tasca la fa l'enllaçador (en anglès, *linker*), que a més permet unir diversos blocs o mòduls d'un mateix programa (projectes) (vegeu la figura 1.5).

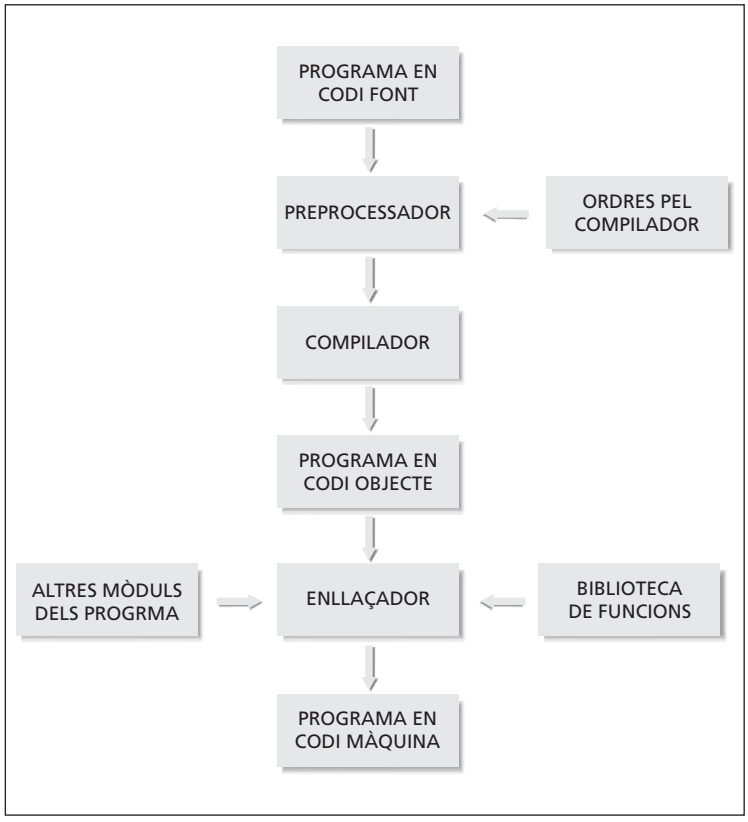


Fig. 1.5 Fases de la confecció d'un programa

Cal destacar que el C++ té una espècie de compilador previ anomenat *preprocessador*, que permet la incorporació d'ordres que faciliten la tasca posterior del compilador i que aniran precedides pel símbol: #.

Les ordres del preprocessador s'anomenen *directrius*.

Així, a l'hora de fer un programa, se'n poden diferenciar les fases següents:

1. *Fase d'edició:* Amb un programa editor de textos (que treballi en codi ASCII), es confecciona un fitxer amb el programa en llenguatge C++, que normalment té extensió *.cpp*. Aquest arxiu conté el codi font del programa.

2. *Fase de traducció:* Amb un programa compilador, l'ordinador tradueix el programa font a programa en binari, que és comprensible directament per l'ordinador. Aquest programa es deixa en un fitxer anomenat *objecte*, amb extensió *.obj*. Posteriorment, se'n fa l'enllaçat i es crea el programa executable amb extensió *.exe*. L'enllaçat és necessari perquè sempre es fa servir arxius externs (de la biblioteca) i en programes sofisticats i de gran volum el procés es realitza en diferents parts i, en última instància, aquestes parts s'han de conjuntar en una única, de manera que tots els arxius ".obj" han d'acabar en un únic arxiu executable.
3. *Fase d'execució i depuració:* S'executa el programa i s'hi detecten possibles funcionaments erronis.

En aquest llibre, es fa servir una eina que ja porta integrats tots els programes necessaris per desenvolupar totes aquestes fases de manera automàtica i eficaç. Es tracta de l'entorn integrat de desenvolupament (IDE), anomenat *DevC/C++*, que és una aplicació que es distribueix amb la llicència GPL.

1.5 Sistemes de numeració

Al llarg de la història, s'han desenvolupat diferents sistemes de numeració, tots ells amb el mateix objectiu: proporcionar una manera pràctica per comptar els objectes. Els primers sistemes de numeració eren els sistemes egipci, grec, xinès, babilònic i maia, els quals eren no-posicionals i no gaire pràctics. A mesura que creixia la necessitat de comptar un nombre més gran d'objectes, es van descobrir nous sistemes de numeració fins a arribar als sistemes de numeració posicionals, entre els quals hi ha el sistema decimal que utilitzem actualment. Així, els sistemes de numeració es classifiquen en posicionals i no posicionals. En els primers, els dígits tenen el valor del símbol utilitzat, que no depèn de la posició que ocupen en el nombre, mentre que en els segons el valor d'un dígit depèn tant del símbol utilitzat com de la posició que aquest símbol ocupa en el nombre. Per exemple, el sistema de numeració egipci és no posicional; en canvi, el babilònic és posicional (vegeu la figura 1.6 i la secció 1.5.5).

Mohammed ibn Musa al Khawarizmi (780-846), matemàtic àrab, va introduir a Occident el sistema de numeració decimal.

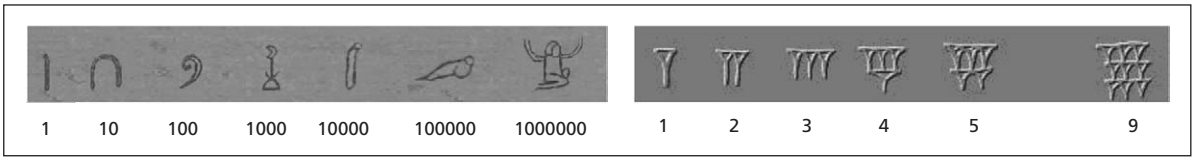


Fig. 1.6 Sistema de numeració egipci (esquerra). Sistema de numeració babilònic (dreta)

Darrere de tot sistema de numeració hi ha el concepte de *base*: es comença a comptar i, quan s'arriba a un nombre determinat, es fa una marca diferent que els representa a tots ells. Precisament, aquest nombre és la base del sistema de numeració. La base que més s'ha utilitzat al llarg de la història és 10, que es correspon amb el nombre de dits amb què comptem. Així, el sistema amb base 10 s'anomena *sistema de numeració decimal*.

No obstant això, és possible definir sistemes de numeració de qualsevol base (nombre natural) ja que, en el fons, un sistema de numeració no és res més que *un conjunt de símbols* i *un conjunt de regles* de generació que permeten construir tots els nombres vàlids en el sistema. Si un sistema de numeració posicional té base b, vol dir dues coses: *a)* que disposem de b símbols diferents per a escriure els possibles nombres, i *b)* que b unitats formen una unitat d'ordre superior (cada b unitats canviem d'ordre).

El sistema sexagesimal (base 60) va ser creat pels babilonis cap a l'any 200 aC.

Un sistema de numeració no és res més que *un conjunt de símbols* i *un conjunt de regles* de generació que permeten construir tots els nombres vàlids en el sistema.

1.5.1 Teorema fonamental de sistemes de numeració

Aquest teorema estableix la manera de representar els nombres d'un sistema posicional de base b qualsevol. Segons aquest teorema, un nombre enter N en un sistema posicional de base b s'expressa amb n xifres (símbols del sistema) de la forma següent:

$$N = d_n d_{n-1} \dots d_0 = d_n * b^n + d_{n-1} * b^{n-1} + \cdots + d_1 b^1 + d_0 * b^0 \tag{1.1}$$

on $d_0, d_1, \dots, d_n$ són símbols permesos en el sistema. Observeu que, en aquesta representació, cada dígit d_i és multiplicat per la potència i-èsima de la base b; per tant, el valor del nombre resultant N depèn tant del valor dels dígits com de les seves posicions dins el nombre –això fa que sigui un sistema de numeració posicional.

En un sistema de numeració posicional, el valor del nombre representat depèn tant del valor dels dígits com de les seves posicions dins el nombre.

El primer que va utilitzar la coma flotant va ser l'astrònom italià Giovanni Magini.

La representació anterior és estesa fàcilment per nombres amb coma flotant usant les potències de b amb exponent negatiu.

$$\begin{aligned} N_{b)} &= d_n d_{n-1} \cdots d_0, r_1 r_2 \cdots r_p = \\ &= d_n * b^n + d_{n-1} * b^{n-1} + \cdots + d_1 b^1 + d_0 * b^0 + r_1 * b^{-1} + r_2 * b^{-2} + \cdots + r_p * b^{-p} \end{aligned}$$

Si bé es poden definir diferents sistemes de numeració per diferents valors de la base b, a la pràctica, pocs sistemes de numeració han resultat ser útils. En concret, hi ha el sistema decimal que fem servir a la vida quotidiana i els sistemes binari, octal i hexadecimal, usats en sistemes d'ordinadors, sistemes digitals etc. per representar eficientment la informació.

El sistema hexadecimal va ser introduït per IBM l'any 1963.

El sistema de numeració decimal és el més usat a la vida quotidiana, mentre que els sistemes binari, octal i hexadecimal són utilitzats en sistemes de computació i digitals.

1.5.2 Sistema decimal

En el sistema decimal, la base és $b = 10$; per tant, hi ha 10 símbols (dígits) que es poden utilitzar per generar nombres del sistema. Aquest conjunt de símbols és $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Així, d'acord amb

l'equació (1.2), qualsevol nombre enter N és representat per:

$$N_{10)} = d_n d_{n-1} \cdots d_0 = d_n * 10^n + d_{n-1} * 10^{n-1} + \cdots + d_1 10^1 + d_0 * 10^0 \quad (1.2)$$

Per exemple, $N = 731 = 7 * 10^2 + 3 * 10^1 + 1 * 10^0$.

Malgrat que el sistema decimal és el més familiar i usat per tothom per fer càlculs, el famós matemàtic francès Laplace va dir que era un error haver escollit 10 com a base del sistema:

> *"La base del nostre sistema de numeració no és divisible entre* 3 *ni entre* 4*, és a dir, entre dos divisors molt utilitzats per la seva senzillesa. La incorporació de dos nous símbols (xifres) hauria donat al sistema de numeració aquest avantatge; però tal innovació seria, sens dubte, contraproduent. Perdríem la utilitat que va donar origen a la nostra aritmètica, que és la possibilitat de calcular amb els dits de les mans."*

Les mesures principals, com ara la longitud, la massa i la capacitat, s'expressen en el sistema decimal, que es basa en múltiples i submúltiples de 10. Així, per passar d'una unitat de mesura a la immediatament inferior cal multiplicar per 10, mentre que per passar d'una unitat de mesura a la immediatament superior cal dividir per 10. Per exemple, $1\ \mathrm{m} = 10\ \mathrm{dm}$.

1.5.3 Sistema binari

El sistema binari s'obté pel valor de la base $b = 2$ i, per tant, conté només dos símbols (xifres) per representar el nombres en aquest sistema: $\{0, 1\}$. En aquest cas, la fórmula de l'equació (1.2), s'escriu per a $b = 2$ de la manera següent:

$$N_{2)} = d_n d_{n-1} \cdots d_0 = d_n * 2^n + d_{n-1} * 2^{n-1} + \cdots + d_1 2^1 + d_0 * 2^0 \quad (1.3)$$

Segons el matemàtic alemany Leibnitz, l'1 representa el Déu i el 0, el no-res!

A partir de l'equació (1.3), tenim la representació següent:

N	$N_{2)}$
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

N	$N_{2)}$
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100
21	10101

N	$N_{2)}$
22	10110
23	10111
24	11000
25	11001
26	11010
27	11011
28	11100
29	11101
30	11110
31	11111
32	100000

Taula 1.1 Representacions binàries dels nombres de zero a trenta-dos

El sistema binari fou proposat per Leibnitz l'any 1679.

Com podem observar a la taula 1.1, tots els nombres són representats com a seqüències de 0 i 1. No obstant això, els valors del 0 i de l'1 depenen de la posició que ocupin en la seqüència.

> El sistema binari usa només dos símbols, el 0 i l'1, per representar els nombres.

De la mateixa manera que el sistema decimal permet representacions de nombres amb part decimal, el sistema binari ho permet fent servir les potències negatives de 2.

Transformar nombres de decimal a binari. Per fer-ho, dividim successivament per dos i anotem les restes. El nombre binari és l'últim quocient, seguit de totes les restes en ordre ascendent (de baix a dalt). Vegem-ne un exemple a continuació:

$167/2 = 83;$ resta $= \mathbf{1}$
$83/2 = 41;$ resta $= \mathbf{1}$
$41/2 = 20;$ resta $= \mathbf{1}$
$20/2 = 10;$ resta $= \mathbf{0}$
$10/2 = 5;$ resta $= \mathbf{0}$
$5/2 = 2;$ resta $= \mathbf{1}$
$2/2 = \mathbf{1};$ resta $= \mathbf{0}$

Llavors, $167_{2)} = 10100111$.

1.5.4 Sistemes octal i hexadecimal

A part del sistema binari, els sistemes més usats en computació són l'octal i l'hexadecimal que s'obtenen amb bases $b = 8$ i $b = 16$, respectivament. Observeu que en tots dos casos es tracta de bases que són potències de 2, de manera que la conversió entre representacions binàries i les octals i hexadecimals són fàcils de dur a terme.

8 bits = 1 byte és la unitat bàsica d'emmagatzemament d'informació.

> Els sistemes octals i decimals són usats en sistemes de computació ja que permeten representacions més compactes que el sistema binari. A més, com que les seves bases són potències de 2, les conversions a binari són immediates.

Sistema octal. En aquest sistema de base 8, tot nombre és representat amb xifres del conjunt $\{0, 1, 2, 3, 4, 5, 6, 7\}$ i la representació segueix la fòrmula:

$$N_{8)} = d_n d_{n-1} \cdots d_0 = d_n * 8^n + d_{n-1} * 8^{n-1} + \cdots + d_1 8^1 + d_0 * 8^0 \tag{1.4}$$

Per transformar un nombre octal en binari, basta transformar en binari cadascuna de les seves xifres i eliminar-ne els possibles zeros de davant. Per exemple: $135_{8)} = 001|011|101$, llavors $135_{8)} = 1011101_{2)}$.

I viceversa, per transformar un nombre binari en octal, basta agrupar els dígits de tres en tres (començant per la dreta) i completar amb 0 a l'esquerra fins a tenir només grups de tres dígits. A cada grup de tres dígits, li podem fer correspondre un dígit octal segons: $000-0$, $001-1$, $010-2$, fins a $111-7$. Per exemple: la representació binària 1000000 s'escriuria: $001|000|000$, que es correspon amb el nombre octal 100.

Sistema hexadecimal. En aquest sistema de base 16, tot nombre és representat amb xifres del conjunt de 16 símbols. Però, com que només disposem de 10 dígits (els de 0 a 9), cal afegir-hi nous símbols, els quals, per convenció, són les 6 lletres A, B, C, D, E, F, que corresponen a $A=10$, $B=11$, $C=12$, $D=13$, $E=14$ i $F=15$. Així doncs, el conjunt de símbols del sistema hexadecimal és $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$. La representació d'un nombre en el sistema hexadecimal segueix la fórmula:

$$N_{16)} = d_n d_{n-1} \cdots d_0 = d_n * 16^n + d_{n-1} * 16^{n-1} + \cdots + d_1 16^1 + d_0 * 16^0 \tag{1.5}$$

AA en hexadecimal equival a 10101010 en binari.

Transformar un nombre hexadecimal en binari és molt fàcil ja que un dígit hexadecimal representa quatre dígits binaris. Per tant, basta transformar en binari cadascuna de les seves xifres i eliminar-ne els possibles zeros de davant. Cal tenir en compte que 0000 és 0, 0001 és 1, 0010 és 2, i així successivament fins a 1111, que és F.

Per exemple: $4A3B_{16)} = 0100|1010|0011|1011$, llavors

$$4A3B_{16)} = 100101000111011_{2)}.$$

I viceversa, per transformar un nombre binari en hexadecimal, basta agrupar els dígits de quatre en quatre (començant per la dreta) i completar amb 0 a l'esquerra fins a tenir només grups de quatre dígits. A cada grup de quatre dígits, li podem fer correspondre un dígit hexadecimal segons la relació donada més amunt.

El sistema hexadecimal s'utilitza per expressar adreces de memòria d'ordinador.

Com es pot apreciar, el sistema hexadecimal permet expressar de manera *compacta* seqüències llargues de 0 i 1. Per exemple, la seqüència:

$$101010111100110111101111$$

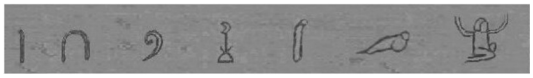
equival a $ABCDEF$ en el sistema hexadecimal.

Aquesta propietat és molt útil per expressar adreces de memòria d'ordinador. Una adreça de memòria és un identificador per a una localització de memòria que permet a un programa d'ordinador emmagatzemar i accedir a una dada. Com que la representació binària és llarga, s'utilitza la representació hexadecimal per a adreces. Per exemple, $0x22ff74$ és l'adreça d'una cel·la de memòria.

$0x22ff74$ és una adreça de memòria d'ordinador.

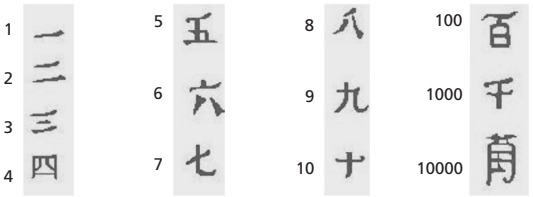
1.5.5 Curiositats: sistemes de numeració antics

Sistema egipci. Els egipcis (tercer mil·lenni abans de Crist) van ser els primers a usar sistemes de numeració per representar nombres en base decimal jeroglífics de la figura.



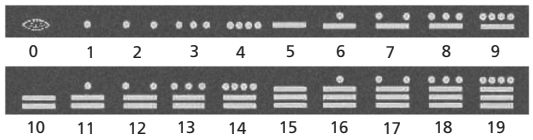
Sistema grec. La civilització grega va utilitzar, cap al 600 aC, el primer sistema de numeració. Es tractava d'un sistema decimal que usava els símbols de la figura.

Sistema xinès. El sistema de numeració xinès es pensa que es va proposar cap al 1500 aC. Aquest sistema té principis del sistema decimal, és a dir, utilitza les unitats i les potències de 10. Utilitza els ideogrames de la figura i la combinació dels nombres fins al deu amb la desena, la centena, el miler i la desena de miler per representar nombres més grans.



Sistema babilònic. Les moltes civilitzacions de l'antiga Mesopotàmia van desenvolupar diversos sistemes de numeració. Així, es va utilitzar un sistema de base 10, additiu fins al 60 i posicional per a nombres superiors. Per a la unitat, s'usava la marca vertical i se'n posaven tantes com fos necessari fins arribar a 10.

Sistema maia. Els maies van destacar per idear un sistema de base 20 amb el 5 com a base auxiliar. La unitat es representava per un punt. Així, dos, tres i quatre punts servien per 2, 3 i 4, respectivament, mentre que 5 era una ratlla horitzontal, a la qual se li afegien els punts necessaris per representar el 6, 7, 8 i 9. En canvi, per al 10 s'usaven dues ratlles, i de la mateixa manera es continuava fins al 20, amb quatre ratlles, etc.



Sistema romà. Aquest sistema es va desenvolupar a l'antiga Roma. És un sistema de numeració no posicional, és a dir, el valor dels símbols no depèn de la posició. El sistema romà feia servir algunes lletres majúscules com a símbols per representar els números: $\mathrm{I} = 1$, $\mathrm{V} = 5$, $\mathrm{X} = 10$, $\mathrm{L} = 50$, $\mathrm{C} = 100$, $\mathrm{D} = 500$, $\mathrm{M} = 1000$. A la figura es pot apreciar l'entrada a la secció LII del Colosseu.

> El sistema romà no té el zero per expressar la no-existència dels elements.

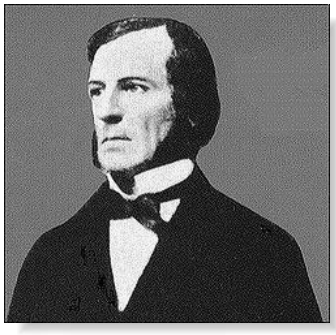
Curiosament, en el sistema romà no existeix el zero, tot i que aquest nombre es coneixia des de l'època babilònica.

1.6 Àlgebra de Boole

El tipus booleà és un dels tipus més usats en informàtica per expressar un estat amb tan sols dos valors: cert o fals. Aquest tipus permet modelar fàcilment estats com ara obert/tancat, on/off, etc. George Boole va definir l'anomenada àlgebra de Boole, que consisteix en una sèrie d'operacions sobre el conjunt de valors {`cert, fals`} ({`true,false`}, en anglès) i permet expressar el resultat d'operacions lògiques com ara comparar valors, definir variables de control en els programes d'ordinadors, etc. Molt especialment, l'àlgebra de Boole és la base del disseny i estudi dels circuits i sistemes digitals.

A l'àlgebra de Boole, $1 + 1 = 1$.

> L'àlgebra de Boole defineix un conjunt d'operacions sobre els valors cert i fals. Malgrat la seva senzillesa, l'àlgebra de Boole és molt potent per expressar i modelar sistemes lògics complexos, com ara els circuits lògics.



George Boole (2 de novembre de 1815-8 de desembre de 1864) fou un matemàtic i filòsof britànic que va inventar l'àlgebra de Boole, la base de l'aritmètica computacional moderna. La importància de l'àlgebra de Boole és que coneix la base de les ciències de la computació. El 1854 va publicar *An Investigation of the Laws of Thought*, on desenvolupava un sistema de regles que li permetia expressar, manipular i simplificar problemes lògics i filosòfics per procediments matemàtics [els arguments només poden tenir dos estats (vertader o fals)].

1.6.1 Operacions booleanes

Sobre el conjunt de valors `true,false`, es defineixen les operacions següents:

- `i`-lògica (`AND`, en anglès). Aquest operador és un operador binari (opera amb dos valors) i es defineix de la manera següent: `a AND b` és cert només si els operands `a` i `b` són tots dos certs. En altres paraules, perquè `a AND b` sigui veritat, cal que `a` i `b` siguin veritats. Qualsevol altra combinació de valors dóna un valor fals.

`AND` és el producte lògic.

- `o`-lògica (`OR`, en anglès). Aquest operador és un operador binari (opera amb dos valors) i es defineix de la manera següent: `a OR b` és fals només si els operands `a` i `b` són tots dos falsos. En altres paraules, perquè `a OR b` sigui una falsedat, cal que `a` i `b` siguin falsos. Qualsevol altra combinació de valors dóna un valor cert.

`OR` és la suma lògica.

- `no`-lògica (`NOT`, en anglès). Aquest operador és un operador unari (opera només amb un valor) i es defineix de la manera següent: `NOT a` dóna el valor contrari de l'operand `a`.

`NOT` és la negació lògica.

A partir d'aquestes definicions, es presenta l'anomenada *taula de veritats* dels operadors booleans:

a	*b*	**NOT** *a*	*a* **AND** *b*	*a* **OR** *b*
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Per analogia amb les operacions aritmètiques de la suma ($+$) i producte ($*$) amb les quals estem familiaritzats, l'operació de la i-lògica s'anomena producte de booleans i la o-lògica s'anomena suma de booleans.

1.6.2 Expressions booleanes

A partir de valors cert i fals, és possible construir expressions més complexes, les anomenades *expressions booleanes*. Aquestes expressions es formen usant operands i operadors, i *sempre* avaluen a cert o fals. Les següents són exemples d'expressions booleanes:

- `x <= y`
- `n < 1000`
- `z >= 0`
- `a=10`

En totes les expressions anteriors, es comparen els valors, i com a resultat s'obté un valor cert o fals depenent dels valors dels operands. Observeu, en particular, que tots els operadors de comparació (també anomenats *operadors relacionals*): $=$, $\neq$, $<$, $<=$, $>$, $>=$ són operadors binaris que donen cert o fals segons si la comparació dels valors dóna cert o fals.

Les expressions es poden combinar usant els operadors booleans per construir altres expressions, a vegades més complexes, com ara:

- `x <= y AND a=10`
- `n < 1000 OR z >= 0`
- `NOT (z >= 0)`

Per avaluar les expressions booleanes, cal seguir les propietats de l'àlgebra de Boole, entre les quals hi ha les anomenades *lleis de Morgan*.

1.6.3 Propietats de l'àlgebra de Boole

Les propietats bàsiques de l'àlgebra de Boole són les següents.

1. La identitat:
   1 **AND** a = a;
   0 **OR** a = a

2. **La commutativitat:**
   a **AND** b = b **AND** a;
   a **OR** b = b **OR** a.

3. **L'associativitat:**
   (a **AND** b) **AND** c = a **AND** (b **AND** c) = a **AND** b **AND** c;
   (a **OR** b) **OR** c = a **OR** (b **OR** c) = a OR b OR c

4. **La distributivitat** de la suma respecte al producte i del producte respecte a la suma:
   a **AND** (b **OR** c) = (a **AND** b) **OR** (a **AND** c);
   a **OR** (b **AND** c) = (a **OR** b) **AND** (a **OR** c).

5. **Lleis de Morgan:**
   NOT(a **AND** b) = (**NOT** a) **OR** (**NOT** b)
   NOT(a **OR** b) = (**NOT** a) **AND** (**NOT** b)

6. **Lleis d'idempotència** de la suma i del producte:
   a **OR** a = a; a **OR NOT** a = 1;
   a **AND** a = a ; a **AND NOT** a = 0

A partir de les operacions bàsiques **AND**, **OR** i **NOT**, es defineixen altres operacions, entre les quals destaquen **NAND** (NOT AND), **NOR** (NOT OR) i **XOR** (OR-EXCLUSIVA).

Nota: Pràcticament tots els llenguatges de programació defineixen un tipus booleà, malgrat que la denominació sigui diferent per a diferents llenguatges. C++ defineix el tipus **bool**.

1.7 Teorema de la divisió entera

Quan es tracta de dividir nombres enters, podem parlar de divisió *entera* i *exacta*, essent la segona un cas particular de la primera. El teorema de la divisió dels enters estableix que, per a qualsevol enter a ($a \neq 0$) i natural b, sempre existeixen dos únics nombres enters q i r tals que es compleix:

$$a = q \cdot b + r, \quad \text{on} \quad 0 \leq r < b$$

La divisió per zero en un programa d'ordinador pot produir un error fatal!

El nombre q rep el nom de *quocient* de la divisió i r rep el nom de *la resta* de la divisió. Observeu que la resta r és estrictament més petita que el valor de b.

En el cas particular de $r = 0$, parlem de divisió exacta. Així, per exemple, per $a = 9$ i $b = 4$, tenim $q = 2$ i $r = 1$ ja que $9 = 2 \cdot 4 + 1$. Mentre que per $a = 18$ i $b = 3$, tenim $q = 6$ i $r = 0$ ja que $18 = 6 \cdot 3$.

El teorema de la divisió entera és vàlid per dos nombres enters qualssevol a i b ($a \neq 0$):

$$a = q \cdot b + r, \quad \text{on} \quad 0 \leq r < |b|$$

Observeu que, en aquest cas, la resta r és estrictament més petita que el valor absolut de b. El teorema de divisió entera permet definir dos operadors: el **div** i el **mod**, que en llenguatges de programació normalment

es denoten pels símbols / i %, respectivament. Usant aquests operadors, és possible fer càlculs sobre les xifres dels nombres enters. Així, si tenim un nombre a (en base decimal), a **mod** 10 ens dóna la darrera xifra del nombre. Per exemple, si $a = 19$, llavors, $19 = 1 \cdot 10 + 9$, o sigui, $r = 9$, que és al mateix temps la darrera xifra del nombre 19, és a dir, $9 = 19$ **mod** 10.

El div i el mod són els operadors de la divisió entera: el primer dóna el quocient de la divisió de dos nombres enters i el segon, la resta de la divisió. En llenguatges de programació, aquests operadors són fonamentals i es denoten pels símbols / i %, respectivament.

1.8 Exercicis proposats

1

Representeu els nombres 1, 10, 100, 1000, 10000, 1000000 en els sistemes binari, octal i hexadecimal. Utilitzeu conversions entre sistemes.

2

Representeu el nombre $146{,}78$ en els sistemes binari, octal ihexadecimal. Utilitzeu conversions entre sistemes.

3

A quins nombres del sistema decimal corresponen les representacions $1100110011_{2)}$, $234_{8)}$ i $7FB5_{16)}$.

4

Digueu quin és el contrari de les afirmacions següents:

- a és més petit que b.
- Plou i fa sol.
- Dues rectes tenen un punt en comú.
- Tots els camins porten a Roma.
- Tots els ànecs són blancs i volen.

5

Feu les taules de veritat de

NOT(a **AND** b **AND** c)

i de

NOT a **OR NOT** b **OR NOT** c.

Quina és la vostra conclusió? Feu el mateix per a

NOT(a **OR** b **OR** c)

i per a

NOT a **AND NOT** b **AND NOT** c.

6

Utilitzeu les operacions div i mod per expressar que un nombre de 3 xifres és capicua (és el mateix nombre si es llegeix d'esquerra a dreta o viceversa). Per exemple, 131 és capicua, però 132 no ho és.

7

Calculeu q i r per a $a = 17$ i $b = 3$, $a = -17$ i $b = 3$, $a = 17$ i $b = -3$, $a = -17$ i $b = -3$.

2 Programació elemental

En aquest capítol, es pretén fer una presentació del llenguatge C++ i alhora introduir els conceptes elementals per tal de poder fer aplicacions bàsiques executables des de la consola. Primerament, es presentaran alguns elements del llenguatge per tal de començar a fer els primers programes, i progressivament s'introduiran l'estructura i la sintaxi pròpia del llenguatge. Per tot això:

El C++ és una evolució del C.

1. Es presenta el concepte de dada i el seu ús, ja sigui com a constant o com a variable, segons el seu tipus.
2. Es presenten i es fan servir les instruccions per introduir dades des del teclat i per mostrar dades a la pantalla.
3. Es presenten les estructures bàsiques de control de flux: seleccions i repeticions, i com construir-les amb les instruccions pròpies del llenguatge.

2.1 El primer programa

S'aprèn a programar programant, i per això sempre es comença amb un exemple, el típic primer programa amb el qual es fa que surti un missatge de benvinguda per pantalla:

Programa 1

Programa que mostra un missatge per pantalla.

```
#include <iostream>
using namespace std;
int  main(void)
{
    cout  << "Hola mon" <<endl; /*Mostra per pantalla el missatge: Hola mon*/
    system("pause");
    return  0;
}
```

L'execució del programa provoca la sortida que es mostra a la figura 2.1.

Les dues primeres línies són:

```
#include <iostream>
using namespace std;
```

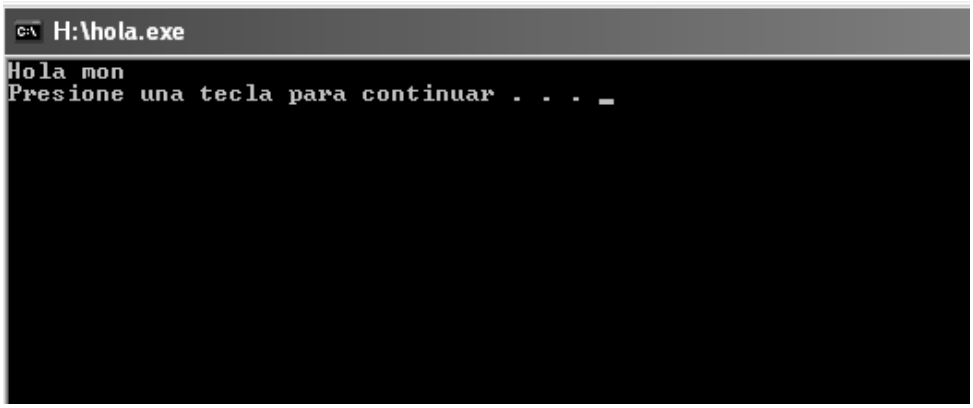


Fig. 2.1 Execució del primer programa

La primera línia indica al compilador que es farà servir alguna funció, en aquest cas es tracta de la funció *cout*, la declaració de la qual es troba a l'arxiu de capçalera *iostream*. D'aquesta manera, el compilador busca l'arxiu *iostream* en un grup estàndard de biblioteques, normalment dintre d'una subcarpeta de les carpetes del DevC/C++. Si, en canvi, es posa: *#include "iostream"*, el compilador buscarà l'arxiu *iostream* en el mateix directori on es trobi l'arxiu amb el codi font.

La segona línia informa al compilador que habiliti un espai de memòria per al nostre programa, de manera que hi hagi una zona concreta per a les variables utilitzades i les components de les biblioteques, l'espai de nom *std*. Això permet un ús més directe de segons quines ordres (no fa falta l'operador de resolució d'àmbit :: propi del C++) i disposar de les funcions de la biblioteca estàndard del C++.

> La instrucció **#include <iostream>** és una ordre per al compilador que permet al programa fer servir les funcions d'entrada i sortida de la biblioteca estàndard. És necessari posar, a més a més, la instrucció: **using namespace std;**.

Cal destacar que el punt i coma final serveix com a indicador de l'acabament d'una instrucció o sentència.

> El punt i coma **;** serveix com a indicador de l'acabament d'una instrucció o sentència.

La tercera línia del programa és:

```
int main (void) {... }
```

que fa referència a la funció *main*. Tot programa ha d'incorporar obligatòriament una funció *main*. És la funció que s'executa primer i que dirigeix l'execució del programa passant les dades a d'altres possibles blocs (funcions) i executant-les. S'ha de tenir present que el C++ considera un programa com un conjunt de funcions la més important de les quals és la funció *main* (per aquesta raó, es diu que el llenguatge C++ és un llenguatge que permet programació estructurada i modular), tal com es veurà al capitol 4. Concretament, la línia: *int main (void)* indica que *main* retorna un valor de tipus sencer (*int*) i que accepta una llista d'arguments buida (*void*), és a dir, que no accepta dades d'entrada i/o sortida. Entre les dues claus ({... }) s'escriuen les instruccions que s'han d'executar a l'hora de dur a terme la funció *main*, cosa que es fa automàticament quan s'executa el programa. Al programa que s'ha presentat hi ha dues instruccions:

```
cout<<''Hola mon''<<endl;
/*Mostra per pantalla el missatge:Hola mon*/
system(''pause'');
return 0;
```

La primera indica que la cadena de caràcters `"Hola mon"` (tot el que sigui una paraula o frase sempre s'ha de posar entre cometes dobles) s'ha d'enviar al flux de sortida estàndard (*cout*), que està associat directament amb la pantalla de l'ordinador. Com que *cout* està definida a l'arxiu de la biblioteca estàndard *iostream*, és per això que s'ha d'incorporar mitjançant la directiva *#include* <*iostream*> (s'anomena l'arxiu de capçalera).

cout es pot entendre com a "console output".

El text entre les barres i els asteriscs (/* i */) és un comentari. Malgrat que el compilador ignora els comentaris, la seva presència és molt important, ja que milloren la legibilitat del codi i el seu seguiment. Es poden fer servir també dues barres (//) per assenyalar l'inici d'un comentari quan aquest ocupa només una línia.

És aconsellable posar comentaris. D'aquesta manera, es documenta el codi i se'n facilita la legibilitat i el seguiment. Els comentaris són ignorats pel compilador. Hi ha principalment dues formes de posar comentaris: amb la doble barra invertida (//), on el comentari és tot el text que va des de // fins al final de la línia, o amb els símbols /* i */ , on el comentari és tot el text que queda entre ells.

La instrucció següent indica que s'ha d'enviar el text: `"Presione una tecla para continuar ...` al flux de sortida estàndard, és a dir, a la pantalla, i s'ha d'esperar que l'usuari premi una tecla per finalitzar l'execució del programa. Aquesta instrucció depèn de l'entorn que es faci servir (sistema operatiu), ja que es tracta de fer una crida al sistema des del programa perquè el propi sistema executi l'ordre posada entre cometes. L'última instrucció és: *return 0*, que indica al sistema operatiu que el programa ha acabat; per defecte, el compilador ja l'afegeix, per la qual cosa no és necessari posar-la.

La instrucció final: return 0, no cal posar-la ja que per defecte el compilador ja ho fa.

Les instruccions d'un programa s'executen seqüencialment, és a dir, una a continuació de l'altra.

Un cop editat el programa, s'ha de compilar. Si tot va bé, el compilador indicarà `"Done"` i, en cas contrari, informarà de l'existència d'errors sintàctics al programa: paraules mal escrites, falta d'un punt i coma, claus desaparellades, etc. S'ha de refer el programa tantes vegades com sigui necessari fins que no hi hagi cap errada d'aquest tipus. Finalment, s'executa el programa i se'n comprova el funcionament correcte. Si s'hi detecta algun problema en el seu funcionament, és que hi ha un error de lògica, amb la qual cosa s'ha de refer el programa novament.

Tot programa en C++ conté sempre la funció *main()*. És en aquesta funció on comença el programa. La definició de la funció *main()* es troba entre els símbols { i }. Aquests són els símbols que delimiten un bloc de sentències.

2.2 Entrada i sortida de dades

Tot programa ha de poder comunicar-se amb el seu entorn, és a dir, ha de poder rebre dades de l'exterior (entrada d'informació) i/o enviar dades a l'exterior (sortida d'informació). S'anomena *flux* de dades (en anglès, *stream*) tot dispositiu que pugui fer de font o destinació de dades. Així, el teclat, la pantalla (el seu conjunt s'anomena *consola*) i els arxius són fluxos de dades. Els fluxos estàndards en C++ són tres funcions declarades a la biblioteca *iostream*:

1. *cout*: és el flux de sortida estàndard associat a la pantalla
2. *cin*: és el flux d'entrada estàndard associat al teclat
3. *cerr*: és el flux d'error estàndard associat a la pantalla

En C++, es parla sovint d'*objectes* en comptes de funcions.

Als fluxos de dades se'ls associen funcions i operadors, com són << i >>, que permeten obtenir dades del flux si és d'entrada, o escriure dades al flux si és de sortida.

cout i *cin* són els substituts dels antics *printf()* i *scanf()*, propis del C.

Finalment, avancem en C++ es poden declarar fluxos addicionals i associar-los amb arxius, i d'aquesta manera es poden llegir i escriure arxius.

Programa 2

Programa que demana el nom i l'edat de l'usuari i mostra una salutació personalitzada.

```
#include <iostream>
using namespace std;

int main(void)
{
string nom;
int edat;
cout <<"Si us plau, escriu el teu nom: " << endl;
cin >> nom;
cout <<"Si us plau, escriu la teva edat: " << endl;
cin >> edat;
cout <<"Hola " <<nom<<" la teva edat es  " <<edat<<" anys" <<endl;
system("pause");
}
```

En aquest exercici, es declaren dues variables: ***string nom;*** i ***int edat;***. La primera, anomenada ***nom***, permet emmagatzemar una cadena de caràcters o una paraula (***string***), i la segona, anomenada ***edat***, permet emmagatzemar un valor enter (***int***).

> En C++, és obligatori declarar totes les variables abans d'usar-les, o, dit d'una altra manera, "avisar de la seva existència". Les variables permeten a l'ordinador emmagatzemar dades a dintre seu, concretament a la memòria i, per tant, la declaració d'una variable serveix perquè l'ordinador reservi la memòria necessària per al seu emmagatzematge.

Després hi ha la instrucció:

```
cout <<"Si us plau, escriu el teu nom: " << endl;
```

que, com ja s'ha vist, serveix per mostrar en pantalla un missatge. Pel seu ús, es pot imaginar que *cout* és la pantalla i que l'operador $<<$, anomenat operador d'inserció, envia informació a l'objecte *cout* (pantalla). L'avantatge d'aquesta forma d'enviar dades per pantalla és que no fa falta indicar quin tipus de dades enviem. Per enviar una variable numèrica, s'escriu:

```
cout << edat;
```

Per mostrar per pantalla diferents tipus de dades, només fa falta fer servir l'operador $<<$ més d'una vegada (concatenació de l'operador):

```
cout<<"Hola "<<nom<<" la teva edat es: "<<edat<<...
```

El manipulador *endl* insereix un caràcter de nova línia ("\n"), és a dir, provoca un salt de línia i, a més, buida el *buffer*.

El mateix passa amb l'objecte *cin*. Aquest representa l'entrada estàndard (el teclat) i va associat amb l'operador d'extracció: $>>$. La sentència:

```
cin >> nom;
```

llegeix del teclat una cadena de caràcters (sense espais) i l'emmagatzema a la variable *nom*, i la sentència:

```
cin >> edat;
```

llegeix un nombre enter i l'emmagatzema a la variable edat.

2.3 Constants, variables i tipus de dades

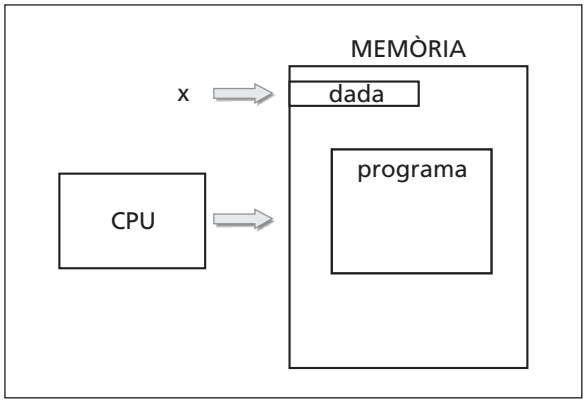


Fig. 2.2 El programa i la variable X ocupen un espai a la memòria de l'ordinador

Com ja s'ha avançat, una variable fa referència a un lloc de la memòria de l'ordinador (figura 2.2). És com si fos una petita capsa o calaix on l'ordinador pot desar dades. Segons com sigui aquesta dada, la capsa haurà de ser més gran o més petita per poder contenir-la. No ocupa el mateix un nombre sencer que un nombre real o un caràcter alfanumèric.

En C++, s'han de declarar –la qual cosa vol dir establir i avisar el compilador de la seva existència– totes les variables que es fan servir en un programa. Per dur a terme la declaració d'una variable, s'ha d'informar el compilador del nom que tindrà i el tipus de dada que contindrà; d'aquesta manera, se'n possibilita la referenciació i se'n fixa la mida.

Un programa ha de declarar totes les variables que fa servir. Això vol dir indicar al principi del programa el tipus de la variable i el seu nom.

Programa 3

Programa que demana el radi d'una circumferència i en calcula la longitud.

```
#include <iostream>
using namespace std;

const float  PI=3.1416;

int main(void) {
float radi;
cout<<"Programa que calcula la longitud d'una circumferencia."<<endl;
cout<<"Si us plau, introduiu el radi: ";
cin >> radi;
cout << "La longitud de la circumferencia es:  " <<PI*radi*2<< endl;
system("pause");

}
```

En aquest exemple, hi ha la instrucció:

```
const float PI=3.1416;
```

que permet definir el valor d'una constant, és a dir, una variable amb un valor fix. D'aquesta manera, un nom queda associat per sempre a un determinat valor. A l'exercici que es presenta es defineix la constant PI i se li associa el valor 3.1416. D'aquesta manera, el programador pot definir les seves pròpies constants.

En el món dels programadors, és una norma consensuada escriure els noms de les constants en majúscules.

Una constant s'ha de declarar segons la sintaxi:

const tipus_dada nom_variable [= valor];

L'ús de constants fa més llegible el codi font d'un programa i les modificacions posteriors que s'hi han de fer.

El llenguatge C++ conté les seves pròpies constants (literals), que formen part de tot ell, i poden ser de diferents tipus, tal com es pot observar a la taula 2.1:

Constants	Descripció
caràcter	Qualsevol caràcter tancat entre cometes simples. Per exemple, el valor 'a'.
nombre sencer	Qualsevol valor enter. Per exemple, els valors: −26 i 32.
nombre fraccionari	Qualsevol valor real. Per exemple, els valors: −1,23 i 0,245.
cadena de caràcters	Qualsevol successió de caràcters (*strings*) delimitada per dobles cometes (" "). Per exemple: "Això és una cadena" .

Taula 2.1 Tipus de constants

Amb la instrucció:

```
float radi;
```

es declara la variable ***radi*** com una variable capaç, és a dir, amb la mida necessària, per emmagatzemar un número fraccionari. Una declaració associa un identificador amb un tipus de dada. El tipus de dada determina com s'ha d'interpretar l'espai de memòria reservat per a cada variable (grandària de la capsa).

> Una *variable* pot canviar el seu valor al llarg de l'execució d'un programa. Una *constant* és una variable que manté un mateix valor prefixat al llarg de l'execució del programa.

Tipus de dada	Descripció
char	Caràcter alfanumèric; s'emmagatzema en un byte
int	Nombre sencer; ocupa quatre bytes
float	Nombre fraccionari; s'emmagatzema en quatre bytes
double	Nombre fraccionari més llarg; s'emmagatzema en vuit bytes
bool	Valor lògic o booleà; el seu domini és: true, false
string	Cadena de caràcters: una paraula o una frase

Taula 2.2 Tipus de dades

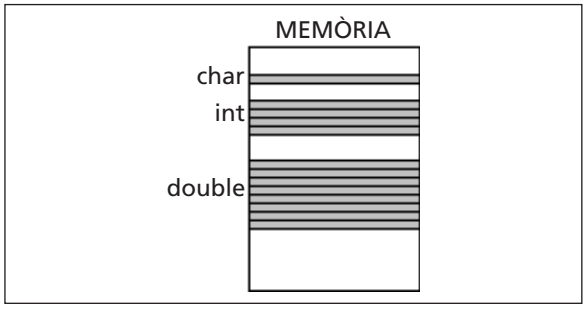


Fig. 2.3 Ocupació de tres tipus de variables a la memòria

Els tipus de dades més comuns, anomenats atòmics, elementals o predefinits, en C++ són els que es defineixen a la taula 2.2.

Segons aquesta taula, es pot comprovar que una variable de tipus ***char*** ocupa 1 byte de memòria, una de tipus ***int*** ocupa quatre vegades més (quatre bytes) i una de tipus ***double*** vuit vegades més (vuit bytes), tal com es representa a la figura 2.3.

El tipus bool i el tipus string són exclusius del C++.

A partir d'aquest tipus de dades se'n constitueixen altres de més complexos, com les taules i les estructures, que es tractaran en capítols posteriors.

> Tota variable s'ha de declarar segons la sintaxi:
>
> **tipus_dada nom_variable [= valor];**
>
> Una declaració comporta, per tant, la introducció d'un nom i l'assignació d'un lloc a la memòria.

"Inicialitzar" una variable és donar un valor de partida a una variable.

Una variable pot tenir de partida un valor inicial (els claudàtors indiquen opcionalitat). Cal destacar que, si una variable no s'inicialitza en el moment de la seva declaració, contindrà un valor indeterminat que

dependrà del que hagi fet l'ordinador en execucions anteriors. Per exemple, en les declaracions següents:

```
double x;
```

La variable x és de tipus fraccionari, amb un valor inicial assignat per la màquina.

```
char lletra = 'a';
```

La variable "lletra" és de tipus `char` i amb valor inicial: 'a'.

Cal destacar que una mateixa declaració pot contenir diversos noms de variables d'un mateix tipus, separats per comes. Així, a la declaració:

```
int n=10, m, p=-7;
```

les variables: n, m i p són variables de tipus enter, la primera amb un valor inicial 10 i la tercera amb un valor inicial -7, mentre que la segona tindrà un valor inicial desconegut per nosaltres i fixat per l'ordinador.

A l'exemple presentat, es demana a l'usuari que introdueixi el valor del radi:

```
cin >> radi;
```

per fer el càlcul de la longitud d'una circumferència amb aquest radi:

```
cout<<"La ...  es:  "<<PI*radi*2<<endl;
```

de manera que es mostra per pantalla el resultat de l'expressió: *PI*radi*2*, on es multiplica (*) el contingut de la variable *radi* pel número 2 i per la constant *PI*.

Programa 4

Programa per fer l'equivalència de polzades a centímetres

```
#include <iostream>
using namespace std;

int main(void)
{
    float pols,cms;

    cout<<"Si us plau, introduiu el nombre de polzades: "<<endl;
    cin>>pols;
    cms=2.54*pols;     //fa el càlcul i assigna a cms el valor calculat
    cout << pols << " polzades equivalen a " << cms <<  " centimetres. ";
    cout << endl << endl;
    system("pause");
}
```

En aquest cas, es fan servir dues variables reals del tipus `float`, anomenades *pols* i *cms*, que s'han declarat mitjançant la instrucció:

```
float pols,cms;
```

El valor de la variable *pols* és llegit des del teclat i és assignat directament amb la sentència *cin*. La variable *cms* és calculada i assignada amb la sentència d'assignació següent:

```
cms=2.54*pols;
```

que fa el càlcul de la conversió de polzades a centímetres i assigna a la variable *cms* la seva equivalència. En aquest cas, l'expressió és el resultat de multiplicar (*) el contingut de la variable *pols* pel número 2.54. A la figura 2.4 es pot veure com varia el contingut de les variables *cms* i *pols* des del moment de la seva declaració, en què el seu valor és indeterminat.

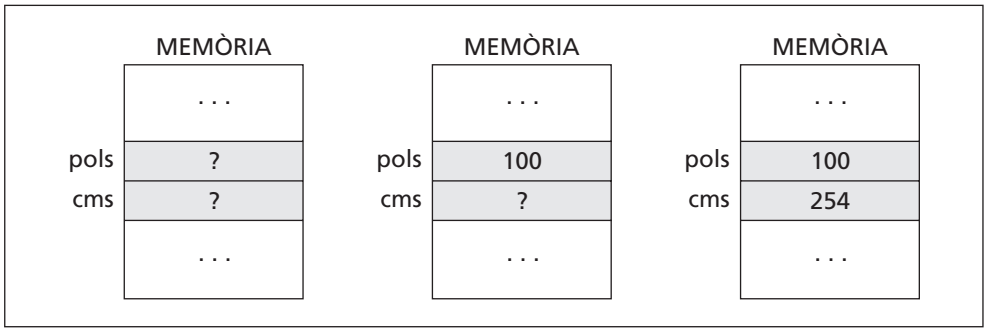


Fig. 2.4 Declaració de les variables *cms* i *pols* (esquerra). Entrada per mitjà del teclat del valor de *cms* : 100 (centre). Valor calculat de la variable *pols* (dreta)

L'operador assignació (=) es fa servir per assignar el valor de l'expressió de la dreta a la variable de l'esquerra.

En C++, el punt entre dos nombres representa el punt decimal de les quantitats reals.
En C++, el símbol = es correspon amb l'operador d'assignació i no amb l'operador d'igualtat.

2.4 Operadors i expressions

Un operador és un símbol alfanumèric que relaciona una o més variables, constants o expressions per dur a terme una operació determinada i, per tant, indica com han de ser processades les dades.

Una expressió és una seqüència d'operadors i operands que especifiquen una operació determinada, per exemple: $10 * (x/100)$. Les expressions poden contenir parèntesis $(...)$ per tal d'agrupar-ne una part i per tal d'alterar la jerarquia de les operacions.

Una expressió és una combinació de variables, constants i operadors.

En C++, hi ha diferents tipus d'operadors, que es presenten tot seguit.

2.4.1 Operadors aritmètics

Són els que permeten fer operacions matemàtiques. Es relacionen a la taula 2.3.

Operador	Descripció
+	Suma
−	Resta
*	Producte
/	Divisió
%	Mòdul (residu de la divisió entera)
−	(Canvi de signe)

Taula 2.3 Operadors aritmètics

2.4.2 Operadors relacionals

Permeten analitzar si es compleixen o no unes determinades condicions. Si una condició es compleix, es diu que és certa (*true*); en cas contrari, es diu que és falsa (*false*). A la taula 2.4, se'n mostren els més usuals.

Operador	Descripció
==	Igual que (no s'ha de confondre amb l'operador d'assignació =)
<	Menor que
>	Major que
<=	Menor o igual que
>=	Major o igual que
! =	Diferent que

Taula 2.4 Operadors relacionals

Per defecte, *false* es representa amb un 0 i *true* amb un 1.

2.4.3 Operadors lògics

Són els que s'utilitzen per realitzar operacions lògiques que permeten combinar els resultats dels operadors relacionals. Es mostren a la taula 2.5.

Operador	Descripció
!	Negació (Torna un 1 si l'operand és 0 o un 0 si l'operand és un 1.)
&&	Conjunció i (AND) (Torna un 1 només en el cas que els dos operands siguin 1.)
||	Disjunció o (OR) (Torna 0 només en el cas que els dos operands siguin 0.)

Taula 2.5 Operadors lògics

2.4.4 Operador d'assignació

Per poder treballar amb els continguts d'una variable, es fan servir instruccions d'assignació.

La sintaxi genèrica d'una instrucció d'assignació és:

nom_variable = expressió;

L'operador d'assignació és (=), que no s'ha de confondre amb una igualtat matemàtica. Ara, el símbol de la igualtat s'ha d'entendre com l'operador mitjançant el qual s'assigna a la variable indicada el resultat d'avaluar l'expressió. Per exemple, amb la instrucció:

```
y = x + (2*z);
```

s'assigna a la variable *y* el valor de l'expressió $x + (2 * z)$.

És molt important entendre el funcionament d'una instrucció d'assignació: *s'avalua l'expressió de la dreta i, un cop se'n determina el valor, aquest es posa dins de la variable indicada ("calaix" de la memòria)*.

Programa 5

Programa que demana el radi d'un cercle i en calcula l'àrea

```
#include <iostream>
using namespace std;

const double PI=3.1416;
int main(void)
{
    double radi;
    cout << "Programa que calcula l'area d'un cercle:"<< endl;
    cout << "Si us plau, introduiu el radi: ";
    cin >> radi;
    cout << "L'area es:  " << PI*radi*radi << endl;
    system("pause");
}
```

En aquest exemple, la instrucció:

```
cout << "L'area es:  " << PI*radi*radi << endl;
```

és equivalent a fer:

```
double resultat;
resultat = PI*radi*radi;
cout << "L'area es: " << resultat <<endl;
```

però amb el preu de fer servir una variable addicional (*resultat*), la qual cosa fa que aquesta no sigui la solució més òptima. D'aquesta manera, es comprova que el valor resultant d'avaluar l'expressió *PI*radi*radi* es posa a la variable *resultat* per mostrar-ne posteriorment el contingut per pantalla. Cal assenyalar que l'expressió: *PI*radi*radi* es pot substituir per *PI*pow(radi,2.0);* on la funció *pow(a,b)* ja està predefinida a la biblioteca de C++ (concretament, a l'arxiu de capçalera *math.h*) i permet fer l'operació potenciació amb la base (radi) i els exponents indicats, sempre que siguin de tipus *double*.

Per fer servir una funció predefinida, per exemple una funció de la biblioteca estàndard, el programa ha de contenir una directiva *#include* amb un arxiu de capçalera on es trobi la definició d'aquesta funció. S'han de conèixer els arxius capçaleres que són necessaris per a cada funció en particular.

Finalment, cal assenyalar que a l'expressió de la dreta pot aparèixer la mateixa variable a la qual es vol assignar el resultat de l'expressió. D'aquesta manera, una sentència com:

```
x = x+1;
```

és totalment correcta en C++, encara que matemàticament sembli incorrecta, ja que el seu efecte és incrementar en un el valor que conté la variable x.

A més de l'operador d'assignació (=), existeixen quatre operadors d'assignació més: (**+=**, **–=**, ***=** i **/=**):

a += b; és equivalent a: $a = a + b$;

a –= b; és equivalent a: $a = a - b$;

a *= b; és equivalent a: $a = a * b$;

a /= b; és equivalent a: $a = a / b$;

2.4.5 Operadors incrementals

Els operadors incrementals **(+ +)** i **(– –)** són operadors unaris (actuen sobre un sol operant) que incrementen o disminueixen respectivament en una unitat el valor de la variable que afecten. Aquests operadors poden anar davant la variable o darrere la variable. A continuació, es mostra la diferència entre aquests dos usos dels operadors incrementals:

```
x = 1;
y = x++;   /* x = 2 i y = 1 */
y = ++x;   /* x = 2 i y = 2 */
```

A la segona assignació, primer s'assigna a y el valor actual de x i, a continuació, s'incrementa la variable x. A la tercera assignació, primer s'incrementa la variable x i, a continuació, s'assigna el seu valor obtingut a la variable y.

Cal destacar l'ús molt comú que es fa de l'expressió:

```
x=x+1;
```

amb la seva expressió equivalent, molt més compacta:

```
x++;
```

L'expressió $x++$ es fa servir sovint quan x és una variable comptadora per tal d'incrementar-la.

2.4.6 Operador conversor de tipus

Les conversions de tipus de dades poden ser *implícites* o *explícites*.

- Conversions *implícites*

  Aquest tipus de conversions tenen lloc a les expressions els operadors de les quals barregen variables i constants de tipus diferents. En aquest cas, la variable o constant de menor rang promociona el rang de l'altra. Per exemple, en les instruccions següents:

```
int x;
double y;
doutble z = x+y;
```

  l'operador suma $(+)$ converteix el valor de x de tipus int a $double$ abans de fer la suma.

  Aquestes conversions es fan de manera automàtica. Per tal de protegir la informació, en general només es permeten conversions dels tipus de menor rang als de major rang: $char < int < float < double$.

  Les assignacions també poden donar lloc a conversions implícites quan el resultat d'una expressió és assignat a una variable. En aquest cas, el valor de l'expressió del costat dret es converteix en el tipus de la variable del costat esquerre. Per exemple, en el cas:

```
double i = 10.5;
int x = i+1;
```

  el resultat de l'expressió $i+1$ és $11{,}5$ i es convertirà a int per poder assignar-lo a la variable x, amb la qual cosa x es quedarà amb el valor 11, és a dir, l'efecte és fer un truncament de la part fraccionària. Cal destacar que aquesta última circumstància és advertida pel compilador.

- Conversions *explícites*

  Totes aquestes conversions poden ser forçades pel programador mitjançant l'ús d'operadors de conversió, que provoquen, en aquest cas, una *conversió explícita*.

En anglès, forçar la conversió d'un tipus a un altre s'anomena *cast* o *casting*.

La sintaxi d'una conversió explícita és:

(tipus) expressió;

on (*tipus)* és el tipus de dada al qual es vol convertir el resultat d'avaluar l'expressió, que pot ser *int, double o char*.

En C++, també es pot fer servir la notació: tipus (expressió); per a les conversions explícites.

Per exemple:

```
double i=10.5;
double x = (int) i;
```

la variable i es forçada a convertir-se a tipus int i posteriorment s'assigna aquest valor a la variable x. Per tant, serà $x = 10{,}0$.

Un exemple molt comú d'aplicació d'aquest tipus de conversió és quan es vol aconseguir treure la part sencera i la part fraccionària d'un nombre real, tal com es presenta a l'exemple següent.

Programa 6

Programa que demana un nombre real i en treu la part sencera i la part fraccionària.

```
#include <iostream>
using namespace std;
int main(void){
    double num;
    system("clear");
    cout << "Introdueix un nombre real: " << endl;
    cin >> num;
    cout << "El nombre introduit es:  " << num << endl;
    cout << "La seva part sencera es: " << (int)num<< endl;
    cout << "La seva part fraccionaria es : " << num - (int)num << endl;
    system("pause");
    return 0;
}
```

En aquest programa, el nombre que es demana a l'usuari (*num*) s'aconsegueix separar en les seves parts sencera i fraccionària gràcies a les expressions (*int*)*num* i *num*-(*int*)*num*, respectivament, que fan ús de l'operador de conversió explícita (*int*) per considerar només la part sencera de la variable considerada. Cal destacar l'ús de la instrucció *system*("*clear*"); mitjançant la qual es fa una crida al comandament del sistema operatiu `"clear"` (*clear screen*), que s'encarrega d'esborrar la pantalla.

2.4.7 Altres operadors

Cal destacar l'operador *sizeof();* que torna la mida del tipus, variable o expressió introduïda com a argument. Per exemple, en el cas:

```
double dada;
cout<<"Mida de dada: "<<sizeof(dada)<<" Bytes"<<endl;
```

s'obté que *dada* ocupa 8 bytes.

- *Prioritat dels operadors*

  Finalment, cal destacar que aquests operadors tenen unes regles de prioritat establertes. De tota manera, tota expressió entre parèntesis sempre s'avalua primer ja que els parèntesis tenen més prioritat, sempre s'avalua des del més intern cap al més extern. La prioritat i l'ordre d'avaluació dels operadors presentats (operadors ordenats de major a menor prioritat) són:

Tipus d'operador	Prioritat	Tipus d'operador	Prioritat
Operadors unaris	− (canvi de signe), !, (tipus)	*Operadors de comparació*	$<$, $<=$, $>$, $>=$
Operadors multiplicatius	$*$, /	*Operadors d'igualtat*	$==$, $!=$
Operadors additius	$+$, − (resta)	*Operadors lògics*	&&, $\|\|$

En cas de trobar la mateixa prioritat, la regla associativa s'aplica d'esquerra a dreta.

Programa 7

Es demana fer un programa que permeti, un cop demanat un nombre a l'usuari, calcular-ne el quadrat, el cub, l'arrel quadrada, l'invers, el logaritme i l'exponencial.

```
#include <iostream>
#include <math.h>
using namespace std;
int main(void)
{
  double x;
  cout << endl << endl;
  cout << "PETITA CALCULADORA CIENTIFICA"<<endl<<endl;
  cout << "Funcions matematiques F(x)" << endl;
  cout << "Introduiu el valor de x: " << endl;
  cin >> x;
  cout << " Quadrat : " << x*x << endl;
  cout << " Cub : " << pow(x,3) << endl;
  cout << " Arrel Quadrada : " << sqrt(x)<< endl;
  cout << " Invers : " << 1/x << endl;
  cout << " Logaritme neperia: " << log(x) << endl;
  cout << " Logaritme decimal: " << log10(x) << endl;
  cout << " Exponencial : " << exp(x) << endl;
  system("pause");
}
```

Assenyaleu l'ús de les funcions matemàtiques: *pow(base,exponent)*, *sqrt(x)*, *log(x)*, *log10(x)*, *exp(x)*, que entre d'altres es troben definides a la biblioteca *math.h*.

2.5 Instruccions de control

Una instrucció o sentència és una ordre executable directament per la màquina, com pot ser una assignació o una operació, i que ha d'acabar necessàriament amb el caràcter **;** (punt i coma). Les instruccions sempre s'executen seqüencialment, és a dir, una darrere l'altra, i per això el punt i coma es pot considerar l'operador de seqüenciació. En molts casos, haurem de fer blocs d'instruccions, és a dir, conjunts d'instruccions consecutives, separades per punt i coma, i posades entre claus {}:

```
{
. .
bloc d'instruccions;
..
}
```

Un ordinador de forma natural executa les instruccions una darrere l'altra, segons l'ordre en què es troben.

El ; (punt i coma) és el delimitador d'instruccions i {} són els delimitadors dels blocs d'instruccions. Tota instrucció en C++, incloses les declaracions de variables, acaben en ;.

Per trencar aquest ordre seqüencial d'executar les instruccions propi dels ordinadors, es disposa de les instruccions *condicionals* i de les instruccions *iteratives*.

2.5.1 Instruccions condicionals: instrucció *if*

En molts casos, es necessita que els programes executin o no, depenent que es compleixi o no una condició, una acció o més. Això s'aconsegueix amb les instruccions condicionals, també anomenades *bifurcacions*.

Les instruccions condicionals més simples de C++ són la instrucció *if* i la instrucció *if...else*.

La sintaxi de la instrucció **if** és:

if (condició) instrucció;
if (condició) instrucció1 else instrucció2;

Una instrucció *if* executa un bloc d'instruccions o un altre depenent del valor d'una condició lògica, que s'ha d'avaluar. En el primer cas, només s'ofereix una alternativa. En el segon cas, hi ha dues opcions: si la condició és certa, s'executa la *instrucció1*, si la condició no és certa, s'executa la *instrucció2*. Per tant, es pot comprovar que la part de l'"*else*" és opcional. A l'exemple següent es fa ús d'aquesta instrucció.

Programa 8

Programa que calcula el màxim de dos nombres.

```
#include <iostream>
using namespace std;
int main(void) {
  int  n1,  n2;
  cout << "Aquest  programa demana  dos  nombres  ";
  cout << "i  calcula  el  maxim."  <<  endl;
  cout << "Si us plau, introduiu  un  nombre:  "<<endl;
  cin >> n1;
  cout << "Si us plau, introduiu  altre  nombre:  "<<endl;;
  cin >> n2;
  if  (n1>n2)
    cout << "El  maxim  es  "  << n1 <<  endl;
  else
    cout << "El  maxim  es  "  << n2 <<  endl;
  system("pause");
}
```

Si, en comptes d'una única instrucció, se'n vol fer més d'una dintre de les opcions establertes, s'han de fer servir les claus delimitadores de bloc:

if (condició) { instrucció11; ... instrucció1n; }
if (condició) { instrucció11; ... instrucció1n;}
else { instrucció21; ... instrucció2n;}

I si, en comptes de fer una tria entre dues possibilitats, es vol disposar de més de dues, s'ha de fer servir l'extensió *else if*:

```
if (condició) { instrucció11; ... instrucció1n;}
else if (condició2) { instrucció21; ... instrucció2n;}
else if (condició3) { instrucció31; ... instrucció3n;}
```

o bé:

```
if (condició1) { instrucció11; ... instrucció1n;}
else if (condició2) { instrucció21; ... instrucció2n;}
else { instrucció31; ... instrucció3n;}
```

En el primer cas, per a les tres opcions hi ha la condició corresponent; en canvi, en el segon, si no es compleixen les dues condicions primeres, acabarem dintre de l'opció *else*, ja que aquesta no té cap condició associada.

D'aquesta manera, es pot ampliar l'exemple anterior de forma que es detecti el cas en què *n1* i *n2* siguin iguals. Llavors, caldrà fer servir aquest tipus d'instrucció:

```
if(n1>n2){
    cout << "El  maxim  es  " << n1 << endl;
}else  if(n2>n1){
    cout << "El  maxim  es  " << n2 << endl;
}else{
    cout << "Els dos nombres  son  iguals." << endl;
}
```

Òbviament, si s'arriba a l'*else* és perquè no es compleixen ni la condició $n1>n2$ ni la condició $n2>n1$, i en aquest cas $n1$ és igual a $n2$.

Un altre exemple d'aplicació d'aquesta instrucció és el programa següent:

Programa 9

Programa que fa la conversió d'euros a pessetes i de pessetes a euros.

```
#include <iostream>
using namespace std;
int main(void) {
   double diners;
   bool euros;
   cout<<"Programa que fa el canvi d'euros a pessetes "<<endl<<endl;
   cout<<"Per passar de pessetes a euros posa un 0, si no un 1: ";
   cin>>euros;
   cout<<endl<<"Introduiu la quantitat que voleu canviar: ";
   cin >> diners;
   cout << endl;
   if (euros) {
       cout<<diners<<" euros son "<<diners*166.386<<" pessetes";
   } else {
       cout<<diners<<" pessetes son "<<diners/166.386 <<" euros";
   }
   cout << endl << endl;
   system("pause");
}
```

El programa fa servir una variable booleana (*euros*), que és la que fa de condició de l'*if* ja que, si *euros* val 1, es farà la conversió en un sentit i, si val 0, en l'altre. Cal assenyalar que es posa *if(euros)*, i això és equivalent a posar *if(euros==1)*.

Les instruccions condicionals es poden encaixar unes dins les altres i donar lloc a estructures condicionals complexes "niades". En cas de tenir més d'una instrucció *if*, una "dintre" de l'altra, cada *else* es refereix a l'*if* més proper.

Per exemple, en el cas:

```
if(n!=0){
  cout << "Numero diferent de zero ";
  if(n>0)
     cout << "i positiu " << endl;
  else
     cout << "i negatiu " << endl;
}else{
     cout << "El numero es zero. " << endl;
}
```

es comprova l'existència d'una instrucció *if* (la que comprova el signe) dintre d'altra més superior, és a dir, feta abans (la que comprova si el nombre donat és zero o no).

2.5.2 Instruccions condicionals: instrucció *switch*

Cal tenir en compte que es disposa d'una altra instrucció condicional que permet triar entre diverses opcions segons quin sigui el resultat d'una expressió. És la instrucció *switch*, que compara successivament el resultat d'avaluar una expressió o el contingut d'una variable amb una llista de constants.

La sintaxi de la instrucció ***switch*** és:
switch (expressió) {
case constant1: instruccions1;
break;
...
case constantn: instruccionsn;
break;
default: instruccions0;
break;
}

En aquesta instrucció, tant l'expressió com les constants han de ser de tipus enter o caràcter. Quan s'avalua l'expressió, si alguna constant, per exemple la *constantn*, coincideix amb el valor de l'expressió, s'executen les instruccions a partir dels dos punts corresponents, és a dir: *instruccionsn* fins a l'ordre *break*. Si cap constant no coincideix amb el valor de l'expressió, s'executen les instruccions del cas *default*, és a dir: *instruccions0*. Cal destacar que la clàusula *default* és opcional.

Per tant, quan s'ha trobat una coincidència, el codi es continua executant fins a trobar una sentència de salt *break*. Aquesta sentència fa que l'execució salti incondicionalment fins al final del bloc de la instrucció *switch*. La instrucció *break* al final de cada cas és molt important, ja que permet sortir de

la instrucció *switch* i fa que no es puguin executar les instruccions associades als casos següents. En determinades circumstàncies, pot interessar justament el contrari, és a dir, que diversos casos puguin compartir el mateix codi. Llavors, no s'haurà de posar el *break* com a delimitador.

Per exemple, en la construcció:

```
switch (nota){
    case 1:
    case 2:
    case 3:
    case 4: cout << "SUSPES" << endl; break;
    case 5: cout << "SUFICIENT" << endl; break;
    case 6: cout << "BE" << endl; break;
    case 7:
    case 8: cout << "NOTABLE" << endl; break;
    case 9:
    case 10: cout << "EXCELLENT" << endl; break;
    default: cout << "Nota incorrecta" << endl; break;
}
```

si la variable *nota* val 1, 2, 3, o 4, apareixerà el missatge SUSPÈS i se sortirà del bloc corresponent del *switch* (aquest és l'efecte de l'ordre *break*; un cop s'ha avaluat *nota*, s'ha de sortir fora del *switch*). Així passarà per a la resta de casos i, en cas que *nota* sigui més gran que 10, sortirà el missatge corresponent a l'opció *default*.

Sovint, la instrucció *switch* es fa servir per a construir menús d'opcions a partir d'ordres donades des del teclat.

Programa 10

Programa que presenta un menú de quatre opcions per escollir.

```
#include <iostream>
using namespace std;
int main(void){
  int opc=1;
  cout << endl << endl;
  cout << "         MENU D'OPCIONS" << endl;
  cout << endl;
  cout << "          1. Opcio 1 " << endl;
  cout << "          2. Opcio 2 " << endl;
  cout << "          3. Opcio 3 " << endl;
  cout << "          4. Sortir " << endl;
  cout << endl;
  cout << "      Seleccioneu una opcio..." ;
  cin >> opc;
  switch(opc){
    case 1:
        cout << "HOLA" << endl;
        break;
    case 2:
        cout << "ADEU" << endl;
        break;
    case 3:
```

```
        cout << "ESPEREU UN MOMENT" << endl;
        break;
    case 4:    //sortida
        break;
    default:;
   }
system("pause");
}
```

Amb les instruccions condicionals o de bifurcació, es pot trencar la seqüencialitat, de manera que s'executi o no una acció o més depenent que es compleixi o no una condició. Les instruccions de selecció que suporta C++ són: *if ... else* i *switch*.

2.5.3 Instruccions iteratives: instrucció *while*

Les estructures repetitives o iteratives són també un element essencial de gairebé tots els llenguatges de programació. Les instruccions iteratives que suporta C++ i que es tracten aquí són: ***while*** i ***for***, les quals permeten repetir un conjunt d'instruccions sempre i que es compleixin unes condicions.

És molt habitual en el món de la programació anomenar *bucle* les estructures repetitives.

La instrucció ***while*** permet executar un bloc d'instruccions sempre que es compleixi una condició lògica.

La sintaxi de la instrucció ***while*** és:
while (condició)
{
// cos del bucle (instruccions)
}

Les instruccions s'executen de la forma següent:

1. S'avalua l'expressió *condició*.
2. Si és certa, s'executen les instruccions del cos del bucle i es torna al punt 1.
3. Si és falsa, s'abandona l'execució del bucle.

Cal destacar que sempre, dintre d'un ***while***, hi ha d'haver alguna instrucció que faci possible que en algun moment no es compleixi la condició ja que si no es provocarà un "bucle infinit".

En un bucle infinit, l'ordinador mai no surt d'ell i queda "penjat".

Per exemple, en el bucle següent l'ordinador queda indefinidament mostrant per pantalla el missatge `"Hola"`, ja que la variable x té el valor 1 de forma permanent:

```
x=1;
while(x==1)
{
    cout <<''Hola''<< endl;
}
```

Una variació de la instrucció *while* és la instrucció *do*:

> La sintaxi de la instrucció **do** és:
> **do**
> {
> **// cos del bucle (instruccions)**
> } **while(condició)**

El procés d'execució és el següent:

1. S'executa el cos del bucle.
2. S'avalua l'expressió *condició*.
3. Si aquesta és falsa, s'abandona el bucle.
4. Si és certa, es torna al punt 1.

El fet diferenciador entre el *while* i el *do* és que en el *do* el cos del bucle s'executa com a mínim un cop, ja que la condició s'avalua al final (a posteriori); en canvi, amb el *while* el primer que es fa és l'avaluació de la condició i, si de principi aquesta ja no es compleix, no es fa el cos del bucle.

Programa 11

Programa que presenta un menú repetitiu amb quatre opcions per escollir i amb la comprovació de l'opció triada.

```
#include <iostream>
using namespace std;
int main(void){
  int opc=1;
  while(opc!=4){
    cout << endl << endl;
    cout << "         MENU D'OPCIONS" << endl;
    cout << endl;
    cout << "          1. Opcio 1 " << endl;
    cout << "          2. Opcio 2 " << endl;
    cout << "          3. Opcio 3 " << endl;
    cout << "          4. Sortir " << endl;
    cout << endl;
    cout << "      Seleccioneu una opcio..." ;
    cin >> opc;
    while((opc<1 || opc>4)) {
      cout << "Opcio no valida" << endl;
      cout << "Torneu a seleccinar una opcio..." << endl;
      cin >> opc;
```

```
    }
    switch(opc){
      case 1:
           cout << "HOLA" << endl;
           break;
      case 2:
           cout << "ADEU" << endl;
           break;
      case 3:
           cout << "ESPEREU UN MOMENT" << endl;
           break;
      case 4:    //sortida
           break;
      default:;
     }
    system("pause");
 }
}
```

Aquest exemple és una millora del exemple 10 en què mitjançant dos bucles fets amb la instrucció *while*, s'aconsegueix, d'una banda, que el menú es mostri de manera repetitiva fins que l'usuari triï l'opció de sortir:

```
while(opc!=4){...}
```

i de l'altra, que es comprovi que s'ha triat una opció de les mostrades:

```
while((opc<1 || opc>4)) {
  cout << "Opcio no valida" << endl;
  cout << "Torneu a seleccionar una opcio..." << endl;
  cin >> opc;
}
```

Programa 12

Programa que, donat un nombre enter positiu, compta les xifres que té i en fa la descomposició en xifres.

```
#include <iostream>
using namespace std;
int main(void) {
   int num, n_xifres=0, copia,xifra;
   cout << "Programa que compta i mostra les xifres d'un numero" << endl;
   cout << "Si us plau, introduiu un numero enter positiu: " << endl;
   cin >> num;
   copia = num;
   if (num==0) {
      n_xifres=1;
   } else {
      while (num!=0) {
         n_xifres++;
         xifra=num%10;
         cout << "Xifra : " << xifra << endl;
         num=num/10;
```

```
        }
    }
    cout << "El numero de xifres de " << copia << " es " << n_xifres << endl;
    system("pause");
}
```

El programa implementa el mètode d'anar dividint el nombre successivament per 10 *(num=num/10;)* fins a obtenir un quocient nul *(while (num!=0))*, de manera que els residus que es van obtenint *(xifra=num%10;)* són les xifres del nombre en ordre invers.

2.5.4 Instruccions iteratives: instrucció *for*

La instrucció *for* executa repetidament un bloc d'instruccions (el cos del *for*) un nombre determinat de vegades, fins que deixa de complir-se una condició determinada.

La sintaxi de la instrucció **for** és:
for (inicial; condició; final)
{
// cos del bucle (instruccions)
}

Les instruccions s'executen de la següent forma:

1. S'executa la instrucció *inicial*.
2. S'avalua l'expressió *condició*.
3. Si aquesta és certa, s'executen les instruccions del cos del bucle, s'executa la instrucció *final* i es torna al punt 2.
4. Si la *condició* és falsa, s'abandona l'execució del bucle.

Normalment, l'execució de la instrucció *final* altera el valor de la condició, de tal forma que en algun moment deixa de complir-se la condició i s'acaba l'execució el bucle.

El cas més normal es fer servir una variable comptadora (ha de ser sempre de tipus enter), a partir de la qual es fa l'avaluació de la condició i l'execució cada vegada de la instrucció *final*, que normalment s'encarrega de fer l'increment d'aquesta variable comptadora. Així, en aquests casos serà:

```
for (var=inicial; var<=final; var=var+increment}
{
instruccions;
}
```

A l'exemple següent es pot comprovar l'ús més habitual d'una instrucció *for*:

Programa 13

Programa que mostra els nombres parells de l'1 al 20.

```
#include <iostream>
using namespace std;
int main(void) {
  int i;
  cout << "Programa que escriu els nombres parells de  l'1 al 20. " << endl;
  for (i=2; i<=20; i=i+2)
 {
  cout << i << endl;
  }
  system("pause");
}
```

La variable comptadora és l'i, que comença en 2 i acabarà justament en 20, i s'anirà incrementant de 2 en 2 $(i = i + 2)$. Per a cada valor de l'i es farà el que hi ha a dintre del *for*, que en aquest cas és precisament mostrar el valor actual en cada moment de la variable i.

La instrucció $i = i + 2$; provoca l'increment de la variable i en 2.

Una altra versió d'aquest programa pot ser:

```
#include <iostream>
using namespace std;
int main(void) {
  int i;
  cout << "Programa que escriu els nombres parells de  l'1 al 20. " << endl;
  for (i=1; i<=20; i++)
  {
      if(i%2==0) {
                 cout << i << endl;
                 }
  }
  system("pause");
}
```

Ara la variable comptadora i comença en 1 i s'incrementa d'un en un, és per això que es posa l'expressió $i++$, que és equivalent a escriure $i = i + 1$, però de forma més compacta.

En C++, s'escriu $i++$, que és equivalent a $i = i + 1$.

Dintre del *for* es col·loca un *if* que fa que es mostri per pantalla el valor de la variable i només quan aquesta sigui parella $(if(i\%2 == 0))$.

A l'exemple següent, posem de manifest l'ús del bucle *for* i de variables de tipus acumulatiu. Aquest tipus de variables són una extensió de l'expressió que ja hem vist per a l'increment d'una variable: $i = i + 1$.

Programa 14

Programa que llegeix una llista de N nombres i en calcula la suma.

```
#include <iostream>
using namespace std;
const int N= 10;
int main(void){
```

```
  double numeroactual, suma=0.0;
  int i;
  cout << "Programa que llegeix una llista de " << N << " numeros " << endl;
  cout << "reals i calcula la seva suma." << endl << endl;
  for (i=0;i<N;i++)
  {
    cout << "Introduiu un numero real: ";
    cin >> numeroactual;
    cout << endl;
    suma = suma + numeroactual;    /*Forma compacta: suma+=numeroactual;*/
  }
  cout << "La suma total dels numeros  es " << suma << endl;
  system("pause");
}
```

Es comprova com a dintre del bucle *for* (que es fa 10 vegades) es demana un número a l'usuari i es va sumant amb el contingut de la variable *suma* (el seu valor inicial ha de ser 0 ja que si no l'ordinador posa un valor indeterminat), que s'actualitza cada cop amb el resultat de cada suma:

```
suma = suma + numeroactual;
```

De forma anàloga, es pot obtenir el càlcul del producte dels 10 nombres introduïts per l'usuari; en aquest cas, en comptes de la variable *suma* es pot fer servir una variable anomenada *producte* i, en comptes de fer la instrucció: *suma = suma + numeroactual* dins del *for*, es posa la instrucció: *producte=producte*numeroactual*. Ara el valor inicial de la variable *producte* ha de ser 1.0, ja que aquest és l'element neutre del producte.

> En C++, es fa servir la notació més compacta: suma+=numeroactual, que és equivalent a l'expressió suma=suma+numeroactual.

Programa 15

Es demana fer un programa que construeixi un triangle com el següent:

$$
\begin{array}{l}
1 \\
1 \;\; 2 \\
1 \;\; 2 \;\; 3 \\
\vdots \\
1 \;\; 2 \;\; 3 \;\; \cdots \;\; n-1 \\
1 \;\; 2 \;\; 3 \;\; \cdots \;\; n-1 \;\; n
\end{array}
$$

on "n" és un nombre demanat a l'usuari.

```
#include <iostream>
using namespace std;
int main() {
int n;
cout << "Programa que construeix un triangle de nombres " << endl;
cout << "Si us plau, introduiu el limit del triangle: ";
cin >> n;
for (int i=1; i <= n; i++) {
```

```
  for (int j=1; j <= i; j++) cout << j << " ";
  cout << endl;
 }
system("pause");
}
```

Cal destacar que tot bucle *for* de la forma:

```
for (inicial; condició; final)
{
 // cos del bucle
}
```

equival a un bucle *while* com el següent:

```
inicial;
while (condició) {
  // cos del bucle
  final;
}
```

> Amb les instruccions de repetició o iteratives, es poden fer bucles, de manera que l'ordinador faci una determinada tasca repetidament depenent que es compleixi o no una condició. Les instruccions de repetició que suporta C++ són: *while, do...while i for*.

2.6 Treballant amb els tipus de dades

Un cop tractades el tipus de dades fonamentals del C++ i les instruccions bàsiques de què es disposa, es pot fer un aprofundiment sobre les característiques dels diferents tipus de dades.

> Els tipus fonamentals de dades del C++ són: *char, int, float, double, bool i string.* Cada tipus ocupa una grandària determinada a la memòria, la qual cosa fa que tingui un rang de valors possibles.

2.6.1 Les variables *char*

Les variables *char* contenen un únic caràcter i s'emmagatzemen en un byte o octet (8 bits). Per tant, una variable *char* pot contenir $2^8 = 256$ valors diferents: de 0 a 255, i és per això que una variable de tipus *char* també es pot tractar com si fos un enter. La correspondència es fa mitjançant el codi ASCII, que és el codi que relaciona tots els caràcters reconeguts per un ordinador de manera biunívoca amb la seva representació numèrica. Per exemple, el caràcter 'A' correspon al nombre enter 65 segons el codi ASCII. Aquesta taula es pot mostrar amb el programa següent:

ASCII és un acrònim de l'anglès *American Standard Code for Information Interchange*.

Programa 16

Programa que mostra el codi ASCII bàsic.

```
#include <iostream>
using namespace std;
int main(void){
    int i;
    cout << endl;
    cout <<"          CODI ASCII            " << endl;
    cout << endl;
    for(i=33;i<128;i++){
      if(i<=99){
        cout << i << "=" << char(i) << "   ";
      }else{
        cout << i << "=" << char(i) << "  ";
      }
      if(i%10==0) cout << endl;
    }
    cout << endl<< endl;
    system("pause");
return 0;
}
```

L'execució del programa provoca la sortida que es mostra a la figura 2.5.

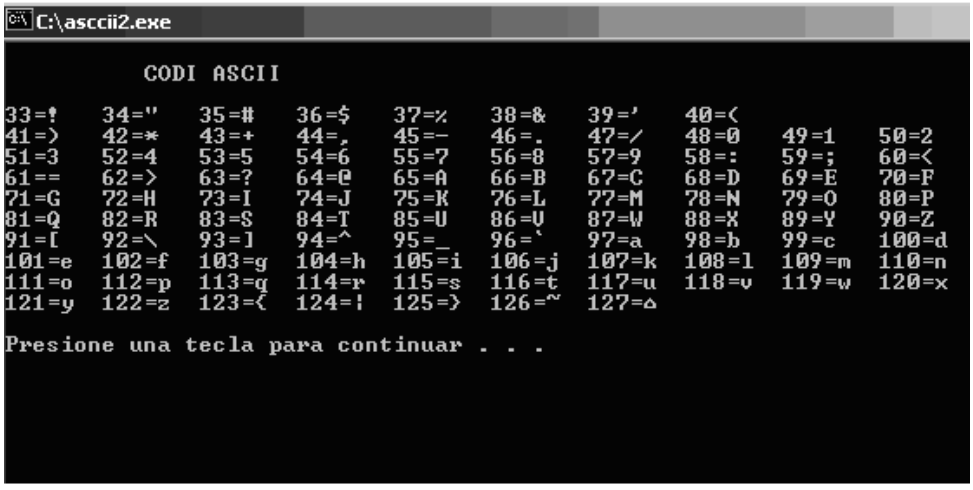


Fig. 2.5 Codi ASCII estàndard

Es comprova com el codi ASCII codifica: els caràcters de control del 0 al 31, els símbols especials del 32 al 47, del 58 al 64 i del 91 al 96, els nombres del 48 al 57, l'alfabet en majúscules del 65 al 91 i en minúscules del 97 al 122.

El codi ASCII estàndard conté 128 representacions i l'ampliat, 256.

Es pot observar que el codi ASCII associa nombres consecutius a les lletres majúscules i minúscules ordenades alfabèticament, la qual cosa facilita les operacions d'ordenació alfabètica. Els caràcters amb codi ASCII més petit que 32 són caràcters especials no imprimibles, s'han evitat mitjançant la variable comptadora del *for* (*i*) que comença en 33. Per exemple, el 32 correspon a un espai en blanc (barra espaiadora), el 13

a l'ENTER i el 27 a l'ESC. D'altra banda, per fer la conversió del valor numèric a la seva representació com a caràcter s'ha fet la conversió explícita: *(char)i*.

Finalment, cal destacar que, donada aquesta dualitat de les variables del tipus *char* que permet considerar-les com a enters, se'ls poden aplicar les mateixes operacions que entre els enters (suma, resta, etc). Es presenta un exemple que fa referència a aquest aspecte:

Programa 17

Programa que mostra com es pot sumar i restar sobre una variable de tipus char.

```
#include <iostream>
using namespace std;
int main(void){
    char c;
    cout << "Introduiu un caracter: " << endl;
    cin >> c;
    cout << "Aquest caracter s'emmagatzema en la variable c. " << endl;
    cout << "La variable 'c' conte la lletra introduida: " << c << endl;
    cout << "El seu valor numeric es ; " << (int)c << endl;
    c=c+2;
    cout << "Si sumem 2 a la variable 'c', conte la lletra: " << c << endl;
    cout << "El seu valor numeric es ; " << (int)c << endl;
    c=c-3;
    cout << "Si restem 3 a la variable 'c', conte la lletra: " << c << endl;
    cout << "El seu valor numeric es ; " << (int)c << endl;
    system("pause");
}
```

Cal destacar que, per llegir un caràcter des del teclat, cal fer algunes consideracions. D'una banda, si es fa amb la instrucció:

```
    cin >> c;
```

s'ha de tenir en compte que aquesta instrucció no llegeix ni l'espai en blanc (barra espaiadora), ni el tabulador, ni els caràcters de control. Si es vol poder llegir algun d'aquests caràcters especials, s'ha de fer servir el modificador de la instrucció *cin*:

```
    cin.get();
```

tal com es pot comprovar amb l'exemple:

Programa 18

Programa que mostra l'ús de cin.get().

```
#include <iostream>
  using namespace std;
  int main(void){
   char tecla='A';
   while(tecla!='x'){
     cout << "Si us plau, premeu una tecla (x per sortir): " <<endl;
     cin >> tecla;  // no agafa l'espai en blanc
    //cin.get(tecla); // detecta l'espai en blanc
```

```
    cout << "La tecla presa en ASCII es : " << (int)tecla;
    cout << " que es correspon amb " << tecla << endl;
  }
 system("pause");
}
```

Amb la instrucció *cin >> tecla;* es mostra per pantalla la tecla presa, però si, per exemple, és l'espai en blanc no fa res. A més, es pot comprovar que el programa no mostra el missatge fins que no es prem la tecla ENTER, és a dir, hem de prémer per exemple: *'a' + ENTER* per veure'n el resultat. Si, en comptes d'aquesta instrucció, es posa la instrucció:

```
cin.get(tecla);
//o el seu equivalent : tecla=cin.get();
```

es pot comprovar com ara sí que es llegeix l'espai en blanc, però també l'ENTER. Això és degut al fet que el teclat té una memòria anomenada *buffer*, des de la qual treballa, i es allà on queden les tecles que es van prement des del teclat. Per no tenir en compte l'ENTER, es pot afegir la instrucció:

```
cin.ignore();
// neteja l'ENTER de després de la lletra
```

justament després de la instrucció *cin.get(tecla);* que precisament ignora l'ultim caràcter llegit. Per comprovar el funcionament del *buffer*, es pot veure que, si es posa una paraula sencera acabada en ENTER, el programa mostra diversos missatges segons les lletres de la paraula introduïda. Per evitar això, es pot fer servir la instrucció:

```
fflush(stdin);
// neteja el buffer del teclat
```

que neteja tot el que hi ha al *buffer* del teclat (inclòs l'ENTER final):

Programa 19

Programa que mostra l'ús fflush(stdin).

```
#include <iostream>
using namespace std;
int main(void){
 char tecla='A';
 while(tecla!='x'){
   fflush(stdin); // neteja el buffer del teclat
   cout << "Si us plau, premeu una tecla (x per sortir): " <<endl;
   cin.get(tecla); // detecta l'espai en blanc
   cout << "La tecla presa en ASCII es : " << int(tecla);
   cout << " que  correspon a " << tecla << endl;
 }
system("pause");
}
```

2.6.2 Les variables *string*

Les variables de tipus **string** són pròpies del C++ i poden contenir tot allò que sigui una *cadena de caràcters*. En realitat, al darrere d'una variable de tipus *string* hi ha un **vector**, tal com es veurà més endavant al

capitol 5, per la qual cosa no és ben bé un tipus bàsic però, a efectes de tractament, el C++ fa posible que es comporti com a tal.

El tipus *string* no existeix en C.

Una variable de tipus ***string*** es pot assignar i comparar tal com es fa també amb una variable de tipus caràcter, només que ara els literals sempre aniran entre comelles dobles ("Això és un ***string***"). A més, els ***strings*** es poden concatenar amb l'operador de la suma $(+)$.

Respecte a la seva lectura i escriptura, amb l'operador $>>$ del ***cin*** es poden llegir ***strings*** sense espais enmig:

```
string s;
cin >> s;
```

Si hi ha espais enmig, s'ha de fer servir la funció ***getline*** amb la qual es llegeix una línia fins a l'ENTER des del teclat i s'assigna a la variable indicada:

```
getline(cin, s);
cout << s;
```

Per a l'escriptura, es fa servir l'operador habitual del ***cout***: $<<$.

Programa 20

Programa que mostra com treballar amb strings.

```
#include <iostream>
using namespace std;

int main(void)
{
  string text1,text2="Hola ",text3=" Que et vagi be!",nom;
  text1 = text2 + text3;
  cout << text1 << endl;
  cout << "Si el teu nom te nomes una paraula, posa: una" << endl;
  cout << "En cas contrari, posa: mes" << endl;
  cin >> text1;
  if(text1=="una")
  {
       cout<< "Posa el teu nom: " << endl;
       cin >> nom;
  }else{
       cout<< "Posa el teu nom: " << endl;
       cin.ignore();
       getline(cin,nom);
  }
  text1=text2+nom+text3;
  cout << text1 << endl;
  cout << "Adeu " << endl;
  system("pause");
 }
```

El programa presentat mostra una salutació personalitzada demanant el nom de l'usuari i llegint-lo, ja sigui d'una sola paraula o de més d'una, i per a això fa diferents usos de quatre variables de tipus *string*: inicialització (*text2 = "Hola", text3 = "Que et vagi bé!"*), concatenació (*text1 = text2 + text3;*), comprovació (*text1 == "una"*) i lectura de més d'una paraula (*getline(cin,nom);*). Cal destacar el *cin.ignore();* previ al *getline*, i que és necessari perquè l'ENTER previ (quan es llegeix la variable *text1*), sigui agafat com a delimitador de la línia a agafar pel *getline*.

2.6.3 Les variables *int*

Les variables del tipus **int** poden ocupar 2 o 4 bytes, depenent de les especificacions de la màquina i la plataforma sobre la qual es treballi. Normalment, sobre Windows XP el rang està comprès entre: $-2.147.483.648$ i $2.147.483.647$, que correspon a una mida de 4 bytes.

2.6.4 Les variables *float* i *double*

Les variables *float* i *double* (també anomenades de punt flotant) permeten representar nombres reals, és a dir, amb una part entera i una part fraccionària. Aquestes variables es representen internament mitjançant la mantissa, que és un nombre comprès entre $0,1$ i $1,0$, i l'exponent, que representa la potència de 10 que multiplica la mantissa. El nombre quedarà definit com: mantissa per 10 elevat a l'exponent. Per exemple, el nombre pi es representa com $0,314592654*10^1$. Tant la mantissa com l'exponent poden ser positius o negatius.

Les variables *float* ocupen 4 bytes (32 bits). Fan servir 24 bits per a la mantissa (1 bit pel signe i 23 pel valor) i 8 bits per a l'exponent (1 bit pel signe i 7 bits pel valor).

Les variables *double* ocupen 8 bytes o 64 bits (53 bits per a la mantissa i 11 per a l'exponent).

A l'exemple següent, es mostren per pantalla, per als diferents tipus de dades, els bytes que ocupen i el rang de dades numèriques que poden contenir. Per aconseguir-ho, es fa servir, d'una banda, l'operador *sizeof()* que torna la mida de la variable o tipus que té com a argument i, de l'altra, les MACROS predefinides dintre de la biblioteca *float.h* per determinar el rang d'un tipus de dada determinat:

Programa 21

Programa que mostra el rang de cada tipus de variable.

```
#include <iostream>
#include <float.h> //MACROS
using namespace std;
int main(void){
  cout << "Mida del tipus char : " << sizeof(char) << " bytes " << endl;
  cout << "Minim: " << CHAR_MIN << endl;
  cout << "Maxim: " << CHAR_MAX << endl;
  cout << endl <<  endl;
  cout << "Mida del tipus int : " << sizeof(int) << " bytes " << endl;
  cout << "Minim: " << INT_MIN << endl;
  cout << "Maxim: " << INT_MAX << endl;
  cout << endl <<  endl;
  cout << "Mida del tipus float : " << sizeof(float) << " bytes" << endl;
  cout << "Minim: " << FLT_MIN << endl;
  cout << "Maxim: " << FLT_MAX << endl;
```

```
  cout << endl <<  endl;
  cout << "Mida del tipus double : " << sizeof(double) << " bytes" << endl;
  cout << "Minim: " << DBL_MIN << endl;
  cout << "Maxim: " << DBL_MAX << endl;
  cout << endl <<  endl;
  system("pause");
}
```

En un ordinador PIV-1,7 GHz amb Windows XP Professional Second Edition provoca la sortida de la figura 2.6, on comprovem la mida dels quatre tipus de dades fonamentals i el seu rang respectiu.

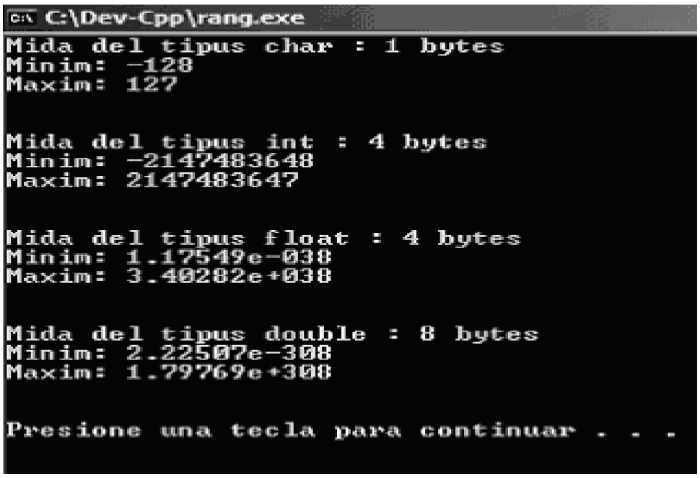


Fig. 2.6 Rangs obtinguts amb el programa presentat

2.6.5 Les variables *bool*

L'estàndard C++ proporciona el tipus de dades del tipus *bool*, que tenen com a valors possibles només *false* o *true*. Aquestes variables es fan servir sovint com una alternativa a utilitzar el 0 per indicar fals i l'1 (diferent de 0) per indicar vertader. Per exemple, amb el programa següent:

Programa 22

Programa que mostra l'ús de les variables booleanes.

```
#include <iostream>
using namespace std;
int main(void)
{
   bool OK;
   cout <<"Si us plau, introduiu un valor per a la variable OK:  " << endl;
   cin >> OK;
   if (OK) //equival a OK==true
     cout<<"Hola"<<endl;
   else
     cout<<"Adeu"<<endl;
  system("pause");
}
```

Es pot comprovar que només amb un 1 mostra el missatge: *"Hola"* i amb qualsevol altre valor que no sigui un 1 mostrarà el missatge: *"Adéu"*.

Un altre exemple d'ús de les variables booleanes i la seva utilitat és el següent:

Programa 23

Programa que comprova si un any és de traspás.

```
#include <iostream>
using namespace std;
int main(void) {
  int any;
  bool bis;
  cout << "Programa que comprova si un any es de traspas," << endl;
  cout << "Si us plau, introduiu un any: " << endl;
  cin >> any;
  bis = ((any%4==0 && any%100!=0) || (any%400==0));
  if (bis) {
     cout << any << " es de  traspas." << endl;
  } else {
     cout << any << " no es de traspas. " << endl;
  }
 system("pause");
}
```

En aquest cas, es vol determinar si un any, introduït per l'usuari i emmagatzemat a la variable ***any***, és de traspàs. La condició que ha de complir és: *"Un any és de traspàs si és divisible per* 4 *però no ho és per* 100*, o en tot cas si és divisible per* 400*"*. Com que això és una condició (si es complirà o no equivalent a 1 o 0), es pot avaluar mitjançant una variable de tipus ***bool***, que en aquest cas s'anomena: ***bis***. A aquesta variable se li assigna el resultat de la condició mitjançant la instrucció:

```
bis=((any%4==0 && any%100!=0) || any%400==0);
```

A la part del parèntesi, es comprova si el residu de dividir ***any*** per 4 és zero (si és així, significarà que ***any*** és divisible per 4) i si, a més (operador && equivalent a operador lògic **AND**), el residu de dividir ***any*** per 100 no és zero (la qual cosa significa que no és divisible per 100); si les dues condicions es compleixen a la vegada ja es pot afirmar que l'any és de traspàs. Però pot haver-hi una altra possibilitat i és que ***any*** sigui divisible per 400, llavors l'any serà de traspàs, i és per això que la unió d'aquesta condició amb les altres dues anteriors es fa amb l'operador lògic **OR** (||). Finalment, segons quin sigui el valor de la variable ***bis*** s'informa per pantalla del resultat de la comprovació (és el que es fa amb ***if(bis)***, que és equivalent a fer ***if(bis==1)***).

Un programa en C++ de moment ha de seguir l'estructura genèrica:

```
// Incorporació dels arxius de capçalera de la llibreria de C++
#include <iostream>
using namespace std;
// Declaració de constants
// Programa principal
int main(void)
{
// Declaració de variables del programa principal
// Cos del programa (instruccions)
system("Pause";) //Provoca una pausa
return 0; // Valor de retorn del programa al S.O.
}
```

En C++, fer ***if(condició)*** equival a fer ***if(condició==1)***.

Programa 24

Realitzeu un programa que demani a l'usuari una marca de cotxes a partir d'un menú d'opcions i en mostri per pantalla la nacionalitat.

[Solució]

A l'hora de confeccionar un programa, es poden seguir els passos següents:

Pas 1. Definició i inicialització de variables Només és necessari disposar d'una variable per demanar a l'usuari l'opció i processar-la. L'anomenen *opcio* i és de tipus *int*, ja que l'usuari ha de teclejar un número d'opció: $1, 2, 3$, o 4.

Pas 2. Cos del programa Primerament, es mostrarà el menú d'opcions. A continuació, s'ha de demanar l'opció a l'usuari i, posteriorment, mitjançant una instrucció *switch*, comparar-la amb els casos possibles: $1, 2, 3$ i 4. Cal destacar que es podria fer igualment amb la instrucció *if*:

```
if(opcio==1)
    cout  <<  "Alemanya"  <<  endl;
else if(opcio==2)
    cout  <<  "Suecia"  << endl;
else if(opcio==3)
    cout <<  "Italia"  << endl;
else if(opcio==4)
    cout  <<  "Espanya"  << endl;
else
    cout  <<  "Opcio incorrecta."  <<  endl;
```

Pas 3. Tractament final Segons quin sigui el resultat de la comparació amb els casos possibles, mostrarem per pantalla el missatge corresponent.

El programa resultant és:

```
#include  <iostream>
using  namespace  std;

int  main()
{
    int  opcio;
    cout  <<  "Trieu el numero de la vostra marca: " << endl;
    cout  <<  "1.  Volkswagen  " << endl;
    cout  <<  "2.  Volvo  " << endl;
    cout  <<  "3.  Ferrari  " << endl;
    cout  <<  "4.  Seat  " << endl;
    cin  >>  opcio;
    switch(opcio)
    {
         case 1: cout << "Alemania" << endl; break;
         case 2: cout << "Suecia" << endl; break;
         case 3: cout << "Italia" << endl; break;
         case 4: cout << "Espana" << endl; break;
         default: cout << "Opcio incorrecta." << endl;
    }
    system("pause");
}
```

Programa 25

Es demana fer un programa que implementi l'algorisme d'Euclides que permet calcular el màxim comú divisor de dos nombres enters positius x i y. L'algorisme consisteix que:

$$\mathrm{mcd}\,(x,y) = \mathrm{mcd}\,(y,r)$$

si $y = 0$ i $r = 0$, on $r = x \bmod y$. Si $x \bmod y = 0$, llavors $\mathrm{mcd}\,(x,y) = y$ i, per a qualsevol altre cas: $\mathrm{mcd}\,(x,y) = \mathrm{mcd}\,(y,r)$.

[Solució]

Només cal disposar de tres variables: x, y i r, de tipus *int* per poder disposar dels dos nombres que cal demanar a l'usuari i el residu corresponent a l'hora de fer la divisió sencera entre ells: *int x, y, r;*. La primera cosa a fer és demanar els dos nombres sencers (x i y) positius; és per això que hi ha un *if* amb la condició: $(x <= 0 \| y <= 0)$. El mètode consisteix a anar agafant el residu (r) de dividir x entre y i, sempre que sigui diferent de zero, fer que els nous valors de x i y siguin el valor de la y i del residu que s'acaba d'obtenir. Com que la condició per aturar el procés és quan s'arriba a un residu de valor 0, abans d'entrar en el *while* s'ha d'obtenir el primer residu i a partir d'ell començar el procés. Cal destacar que l'última instrucció del cos del *while* és precisament obtenir el nou residu amb els nous valors de la x i de la y, la qual cosa farà que amb tota seguretat s'arribi a l'aturada del bucle ja que en algun moment s'arribarà a $r = 0$. Finalment, se'n mostra el resultat final, és a dir, el contingut de la variable y.

El programa resultant és:

```
#include <iostream>
using namespace std;
int main(void) {
   int x, y, r;
   cout << "Programa que implementa l'algorisme d'Euclides " << endl;
   cout << "Introduiu el primer nombre: " << endl;
   cin >> x;
   cout << "Introduiu el segon nombre: " << endl;
   cin >> y;
   if (x<=0 || y<=0) {
     cout << "Els nombres han de ser enters positius." << endl;
   } else {
     r=x%y;
     while (r!=0) {
        x=y;
        y=r;
        r=x%y;
     }
   }
   cout << "El maxim comu divisor es " << y << endl;
   system("pause");
}
```

Programa 26

Es vol realitzar un programa perquè l'usuari pugui jugar amb l'ordinador, de manera que hagi d'encertar un número triat a l'atzar per l'ordinador. Per aconseguir-ho, l'usuari tindrà cinc intents. A cada intent, l'ordinador demanarà a l'usuari un número i ha d'informar si és el número triat i, per tant, si ha encertat. En cas contrari, l'ordinador ha de donar una pista segons si el número introduït per l'usuari és major o menor que el triat per l'ordinador.

En aquest cas, necessitem quatre variables, tres de tipus *int* i una de tipus *bool*, per poder emmagatzemar, respectivament:

1. El número triat per l'ordinador: *int num;*
2. El número que prova en cada intent l'usuari: *int intent;*
3. El nombre d'intents que fa l'usuari: *int i;*
4. Si l'usuari ha encertat el número: *bool no_encertat = true;*

Donat que partim de la situació inicial que no s'ha encertat el número i no s'ha fet cap intent, inicialitzarem les variables corresponents:

```
bool  no_encertat  =  true;
int  num, intent, i  =  0;
```

La primera tasca a fer és que l'ordinador triï un número l'atzar, per la qual cosa és necessari fer servir dues funcions de la llibreria estàndard: **srand()** i **rand()**, incloses a l'arxiu *iostream*. Amb la instrucció **srand(time(NULL));** s'inicialitza el generador de números aleatoris per part de l'ordinador, de manera que aquest posa en marxa el seu rellotge intern justament en començar l'execució del programa a aquest efecte. Amb la funció **rand();** es torna un número triat a l'atzar, que es diposita a la variable *num*, a la qual després s'aplica l'operació *num%11;* perquè estigui comprès entre 0 i 10. Un cop fet això, el pas següent és construir un bucle per poder demanar a l'usuari el número a provar fins a un nombre de vegades no superior a 5 (constant *NUM_INTENTS*) o fins que encerti el número o, dit d'una altra manera, mentre que *no_encertat* sigui *true*:

```
while(i<NUM_INTENTS&&no_encertat)
```

Dintre del *while*, avaluem els diferents casos amb una instrucció *if*, per acabar fent l'increment de la variable comptadora d'intents justament abans de tornar a l'inici del *while(i++)*. Només en el cas que l'usuari encerti el número s'ha de mostrar el missatge corresponent, la qual cosa es fa mitjançant l'avaluació de la variable *no_encertat*. (Cal destacar que *if(no_encertat)* és equivalent a fer: *if(no_encertat==true).*)

El programa resultant és:

```
#include  <iostream>
using  namespace  std;

#define  NUM_INTENTS  5

int  main()  {
bool  no_encertat  =  true;
int  num, intent, i  =  0;
cout  <<  "Endevineu  el  nombre  que  he  triat  entre  0  i  10.";
cout  <<  "Teniu  5  intents."<<  endl;
srand(time(NULL));  //inicialitzador de nombres aleatoris
num  =  rand();
num  =  num%11;//Reduïm  el  rang  del  número  triat  entre  0  i  10.

while(i  <  NUM_INTENTS  &&  no_encertat)
{
cout  <<  "Proveu sort.  Introduiu  un  nombre:  ";
cin  >>  intent;
if  (intent  ==  num)  {
    cout  <<  "OK!!"  <<  endl;
```

```
    no_encertat = false;
}
else if (num < intent)
    cout <<"Nombre introduit mes gran que el que he triat"<< endl;

else
cout<<"Nombre introduit mes petit que el que he triat"<< endl;

i++;
}
if(no_encertat)
cout << "El nombre es el "<< num <<" HEU PERDUT!";
cout << endl;
system("pause");
}
```

Programa 27

Realitzeu un programa que demani a l'usuari dos punts: $(x1,y1)$ i $(x2,y2)$ que formin un rectangle, on $(x1,y1)$ sigui el vèrtex superior de l'esquerra i $(x2,y2)$ el vèrtex inferior de la dreta, essent els dos costats paral·lels als eixos de coordenades. El programa ha de comprovar si un punt demanat a l'usuari està dintre de l'àrea del rectangle o no.

[Solució]

Per poder fer aquest programa, només es necessiten sis variables per poder enmagatzemar les sis coordenades corresponents als tres punts que s'han de demanar a l'usuari: $(x1,y1)$, $(x2,y2)$ i (x,y). Per donar la màxima flexibilitat al programa, declarem aquestes variables de tipus *double* i així podem considerar coordenades fraccionàries:

```
    double x1, y1, x2, y2, x, y;
```

Primerament, s'han de demanar les coordenades dels punts que han de formar el rectangle i tot seguit s'ha de comprovar que efectivament poden constituir un rectangle, per la qual cosa s'ha de comprovar que el punt $(x1,y1)$ estigui per damunt i més a l'esquerra que el punt $(x2,y2)$, és a dir:

```
    if(x1<x2 && y1>y2)
```

Només en cas que sigui així caldrà demanar a l'usuari el punt a comprovar $((x,y))$ i fer la comprovació de si realment el punt està dintre del rectangle en qüestió o no:

```
if((x1<=x && x<=x2) &&  (y1>=y && y>=y2))
```

Segons quina sigui l'avaluació de les condicions de les dues instruccions *if* es mostrarà un missatge o un altre. Si no es compleix el primer *if*, els punts no formaran un rectangle i, si no es compleix el segon *if*, el punt (x,y) estarà fora del rectangle.

El programa resultant és:

```
#include <iostream>
using namespace std;
int main() {
double x1, y1, x2, y2, x, y;
cout << "Es requereixen dos punts:(x1,y1) i (x2,y2) que formin un rectangle" << endl;
```

```
cout <<   "Introduiu el vertex superior de l'esquerra (x1,y1) : " << endl;
cout <<   "Introduiu la coordenada x1: " ;
cin  >>  x1;
cout <<   "Introduuiu la coordenada y1: " ;
cin  >>  y1;
cout  <<  "Introducir  el vertex inferior de la dreta (x2,y2) : " <<  endl;;
cout <<   "Introduiu la coordenada x2: " ;
cin  >>  x2;
cout <<   "Introduiu la coordenada y2: " ;
cin  >>  y2;
if(x1  <  x2  &&  y1  >  y2){
       cout  <<  "Introduiu  les coordenades  del  punt a comprovar : " << endl;
       cout <<   "Introduiu la coordenada x: " ;
       cin  >>  x;
       cout <<   "Introduiu la coordenada y: " ;
       cin  >>  y;
       if((x1  <=  x  &&  x  <=  x2)  &&  (y1  >=  y  &&  y  >=  y2)){
              cout  <<  "El  punt  ("  <<  x  <<  ","  <<  y  <<")  esta dins  del
  rectangle."  <<  endl;
       }else{
            cout  <<  "El  punt ("  <<  x  <<  ","  <<  y  <<") esta  fora  del
  rectangle."  <<  endl;
       }
}else{
cout  <<  "Els  punts  son  incorrectes." << endl;
cout  <<  "El  punt  superior  de l'esquerra ha d'estar ";
cout  << "a l'esquerra i per damunt del punt inferior de la dreta." <<  endl;
}
system("pause");
}
```

Programa 28

Es vol realitzar un programa que demani les coordenades de dos punts: $(x1,y1)$ i $(x2,y2)$ que formen una recta. El programa ha de ser capaç de comprovar si un punt demanat a l'usuari pertany a la recta que passa pels punts donats inicialment.

[Solució]

En aquest cas, es necessiten sis variables de tipus ***double*** per poder disposar de les coordenades dels tres punts a demanar (els dos que constituiran la recta i el que s'ha de comprovar si pertany a la recta), quatre també de tipus ***double*** per poder construir l'equació de la recta, i una variable de tipus ***bool*** per saber si el punt pertany a la recta o no, un cop feta la comprovació.

Primerament, es demanaran les coordenades dels punts que han de formar la recta: $(x1,y1)$ i $(x2,y2)$ i a continuació es comprovarà si efectivament aquests dos punts introduïts formen una recta ja que pot donar-se el cas que:

1. Siguin el mateix punt: *if(x1==x2&&y1==y2)*, per la qual cosa no formaran una recta.
2. La recta sigui paral·lela a l'eix Y: *if(x1 == x2 && y1 != y2)* i, per tant, en fem un tractament a part, ja que l'equació de la recta serà $x = x1$ i el seu pendent serà infinit, cosa que ens donaria problemes si en féssim el tractament directe mitjançant la fórmula general.

En cas de no donar-se cap dels casos anteriors, el programa farà la construcció de l'equació de la recta a partir dels dos punts donats i, un cop demanat el punt a comprovar (x,y), comprovarà si pertany a la recta, cosa que es fa amb la instrucció:

$$dintre = (y == ((numerador\_a * x)/denominador\_a) + \\ +(numerador\_b/denominador\_b));$$

que ens diu si la y compleix o no l'equació i posa a dintre de la variable ***dintre*** el resultat d'aquesta comprovació (***true***, si es compleix, o ***false***, si no es compleix). Finalment, mitjançant l'avaluació de la variable ***dintre***, es mostra si el punt (x,y) pertany o no la recta.

El programa resultant és:

```
#include <iostream>
using namespace std;


int main() {
double x1, y1, x2, y2,x,y;
double numerador_a,denominador_a,numerador_b,denominador_b;
bool dintre;


cout << "Es demanen dos punts (x1,y1) i (x2,y2)" << endl;
cout <<"Introduiu les coordenades del primer punt: "<<endl;
cin >> x1;
cin >> y1;
cout <<"Introduiu les coordenades del segon punt: "<<endl;
cin >> x2;
cin >> y2;
//Es comprova que formin una recta
if (x1 == x2 && y1 == y2) {
    cout << "Son el mateix punt: no formen una recta."<< endl;
}else if(x1 == x2 && y1 != y2) {
//La recta és paral·lela a eix Y, amb l'equació: x=x1
      cout << "Introduiu les coordenades del punt a comprovar:" << endl;
      cin >> x;
      cin >> y;
      if (x==x1) {
          cout<<"El punt ("<< x << "," << y << ") pertany";
          cout << " a la recta X = "<< x1;
      }else {
          cout<<"El punt ("<< x << "," << y << ") no pertany";
          cout << " a la recta X = "<< x1;
          }
}else{
//Càlcul d'a i b de l'equació de la recta: y=ax+b;
numerador_a = y2 - y1;
denominador_a = x2 - x1;
numerador_b =(denominador_a*y1)-(numerador_a*x1);
denominador_b = denominador_a;
cout << "Introduiu coordenades del punt a comprovar." << endl;
cin >> x;
```

```
cin >> y;
dintre = (y==((numerador_a*x)/denominador_a)+(numerador_b/denominador_b));
if (dintre) {
cout<<"El punt ("<< x << "," << y << ") pertany";
cout<<" a la recta: y="<<numerador_a/denominador_a<<"x + ";
cout <<numerador_b/denominador_b<< endl;
} else {
cout<<"El punt("<< x << "," << y << ") no pertany";
cout<<" a la recta: y="<<numerador_a/denominador_a<< "x + ";
cout << numerador_b/denominador_b<< endl;
}
}
system("pause");
}
```

Programa 29

Es tracta de fer un programa per resoldre una equació de segon grau:

$$ax2 + bx + c = 0$$

per a tots els seus casos possibles.

[Solució]

D'una banda, el programa ha de demanar els tres coeficients a, b i c que seran del tipus *double*. Per altra banda, el programa ha de calcular les dues solucions possibles x i y, que també seran de tipus *double*. Finalment, necessitarem una altra variable per poder calcular i avaluar el discriminant, que anomenarem *disc* i que també serà de tipus *double*. Cal destacar que, per fer l'arrel quadrada, es fa servir la funció **sqrt(num)** i, per elevar un nombre a un altre nombre, es fa servir la funció **pow(base,exp)**. Totes dues funcions requereixen paràmetres de tipus *double*, i proporcionen el resultat també de tipus *double*. Per utilitzar aquestes funcions, és necessari incloure al principi del codi l'arxiu de capçalera *math.h*.

Primerament, s'han de demanar els tres coeficients a l'usuari, i tot seguit determinar-ne el discriminant, ja que d'ell dependrà la naturalesa de les solucions de l'equació: *disc=pow(b,2)-4*a*c;*. A partir d'aquest moment, i amb diferents condicions, es poden diferenciar cadascuna de les possibles solucions segons siguin els coeficients a i b ($a==0$ && $b==0$ provoca una equació nul·la i $a==0$ && $b!=0$ provoca una equació de primer grau) i el signe del discriminant ($disc>0$ provoca una equació amb solucions reals, $disc=0$ provoca una equació amb solució real doble i $disc<0$ provoca una equació amb solucions complexes).

El programa resultant és:

```
#include <iostream>
#include <math.h>
using namespace std;
int main(void) {
   double a, b, c, x, y, disc;
   cout << "PROGRAMA PER RESOLDRE EQUACIONS DE SEGON GRAU" << endl;
   cout << "Si us plau, introduiu els coeficients de l'equacio:" << endl;
   cout << "Introduiu el coeficient de segon grau: " ;
   cin >> a;
   cout << "Introduiu el coeficient de primer grau: " ;
   cin >> b;
```

```
    cout << "Introduiu el coeficient independent: " ;
    cin >> c;
    disc=pow(b,2)-4*a*c;

    if (a==0 && b==0) {
        cout << "No es una equacio." << endl;
    } else if (a==0 && b!=0) {
        x=-c/b;
        cout << "Es una equacio de primer grau amb solucio: " << x << endl;
    } else if (disc>0) {
        disc=sqrt(disc);
        x=(-b+disc)/(2*a);
        y=(-b-disc)/(2*a);
        cout << "Hi ha dues solucions reals diferents " << endl;
        cout << " x1 = " <<  x << endl;
        cout << " x2 = " <<  y << endl;
    } else if (disc==0) {
        x=-b/(2*a);
        cout << "Hi ha una solucio doble real:  " << x << endl;
    } else {
        x=-b/(2*a);
        y=(sqrt(-disc)/(2*a));
        if(y<0) y=-y;
        if (y!=1) {
             cout << "Hi ha dues solucions complexes " << endl;
             cout << " x1 = " << x << "+" << y << "i" << endl;
             cout << " x2 = " << x << " << y << "i" << endl;
        } else {
             cout << "Hi ha dues arrels complexes " <<
             cout << " x1 = " << x << "+i" << endl;
             cout << " x2 = " << x << "-i" << endl;
        }
    }

    system("pause");
}
```

2.8 Exercicis proposats

2.8.1 Exercicis bàsics

1

Feu un programa que demani la longitud de la base i de l'alçada d'un rectangle i en calculi l'àrea.

2

Feu un programa que, donada una quantitat de temps en segons, faci la seva descomposició en dies, hores, minuts i segons.

3

Feu un programa que incrementi en un segon una quantitat de temps expressada en dies, hores, minuts i segons.

4

Feu un programa que calculi el factorial d'un nombre enter demanat a l'usuari.

5

Feu un programa que calculi la suma de les xifres parelles d'un número i el producte de les seves xifres senars.

6

Feu un programa que, mitjançant un menú, permeti fer les quatre operacions algebraiques bàsiques amb dos operands.

2.8.2 Exercicis avançats

1

Donades les funcions $f(x,y)$ i $g(x,y)$ tals que:

$$f(x,y) = \begin{cases} 10 & \text{si} \quad x \text{ i } y < 1 \\ 2 & \text{si} \quad x \text{ i } y \geq 1 \end{cases}$$

$$g(x,y) = \begin{cases} 15 & \text{si} \quad x \text{ i } y < 0{,}5 \\ 1 & \text{si} \quad x \text{ i } y \geq 1 \end{cases}$$

per a x i y reals, realitzeu un programa que calculi el valor de la suma de les dues funcions per a un parell de valors (a,b) introduïts per l'usuari.

2

La taxa anual equivalent representa el percentatge real d'interès que s'aplica per una operació de préstec. Sense tenir en compte les despeses, per a una taxa d'interès anual "i" i amb un nombre de "n" pagaments anuals, el TAE es pot calcular mitjançant la fórmula:

$$TAE = (1 + i/n)n - 1$$

Així, amb un interès del 5% anual ($i = 0{,}05$) i amb quotes mensuals ($n = 12$), es tindrà un TAE del $5{,}12$: $(TAE = (1 + 0{,}05/12)12 - 1 = 0{,}0512)$.

Es demana realitzar un programa que, a partir dels valors demanats a l'usuari: nombre de pagaments a l'any (n) i interès anual (i), calculi i mostri per pantalla el TAE corresponent.

3

L'algorisme rus de la multiplicació permet calcular el producte de dos nombres enters. Es tracta d'escriure els dos nombres un al costat de l'altre i anar calculant el doble del primer i la divisió sencera per dos del segon de manera successiva i posant els resultats uns sota els altres per a cada iteració en forma de columnes.

El càlcul acabarà en obtenir un 1 a la columna de les divisions. Per exemple, per multiplicar 48 i 27 es fa:

48	27
96	13
192	6
384	3
768	1

A continuació, per determinar-ne el producte, se sumen els nombres de la columna esquerra que tinguin un nombre company a la dreta parell. En el nostre cas:

48	27
96	13
384	3
768	1

La suma dels nombres que queden de la primera columna serà el resultat del producte:

$$48 + 96 + 384 + 768 = 1296 (= 48 \cdot 27)$$

Es demana realitzar un programa que implementi aquest mètode per tal d'obtenir el resultat de multiplicar dos nombres enters introduïts per l'usuari sense fer servir l'operació de multiplicació.

4

Es diu que dos nombres enters i positius "són amics" si són diferents i la suma dels divisors propis (sense considerar el propi número) del primer és igual a la suma dels divisors propis del segon. Per exemple, els números 33 i 16 són amics, ja que:

Divisors del 33:	1, 3, 11
Suma dels divisors de 33:	15
Divisors del 16:	1, 2, 4, 8
Suma dels divisors del 16:	15

Realitzeu un programa que demani a l'usuari un nombre enter positiu i mostri un amic seu, de manera que el programa acabi en el moment en què en trobi un.

5

Els elements d'una successió geomètrica es poden obtenir a partir de la fórmula següent:

$$a_n = a \cdot r^n$$

on n és l'element n-èsim de la progressió i r la seva raó. Així, per $a = 2$ i $r = 2$, serà $a_1 = 2 \cdot 2^1 = 4$ i $a_4 = 2 \cdot 2^4 = 32$.

D'altra banda, la suma dels n primers termes d'una successió geomètrica es pot calcular amb la fórmula:

$$S_n = a_1 \cdot \frac{r^n - 1}{r - 1}$$

on a_1 és el primer terme, r la raó i n el nombre de termes de la successió geomètrica.

Es demana fer un programa que demani a l'usuari els valors d'a, r (de tipus *double*) i n, (de tipus *int*), mostri per pantalla els n primers termes de la successió a_n i calculi la suma d'aquests n primers termes fent la suma de tots els termes, i comprovi la validesa de la fórmula donada de la suma dels n primers termes d'una progressió geomètrica.

6

Es demana un programa que mostri per pantalla el menú següent:

1. Passar de graus a radians.
2. Passar de radians a graus.
3. Calcular la funció sinus.
4. Calcular la funció cosinus.
5. Sortir del programa.

I faci, segons l'opció triada per l'usuari, el càlcul corresponent.

Nota: Les funcions trigonomètriques es troben definides a l'arxiu *math.h*, de manera que han de treballar amb variables tipus *double* i a partir de l'angle en radians: *double cos(double); double sin(double);*

7

Es demana implementar un programa per resoldre un sistema d'equacions lineals del tipus:

$$\begin{aligned} ax + by &= c \\ dx + ey &= f \end{aligned}$$

on s'han de demanar a l'usuari els sis coeficients i, a partir d'ells, i tenint en compte les fórmules:

$$x = \frac{ce - bf}{ae - bd} \qquad y = \frac{af - cd}{ae - bd}$$

mostrar per pantalla les dues solucions possibles, fent prèviament una discussió del sistema.

3 Seqüències

3.1 Introducció

Als capítols anteriors, s'han presentat els conceptes fonamentals de la programació. Concretament, s'han descrit els tipus dels objectes i les operacions associades, com també les tres estructures bàsiques per realitzar programes: seqüencial, condicional i repetitiva. En aquest capítol, aprendrem a utilitzar aquests conceptes per fer programes d'una certa complexitat i ens centrarem, d'una manera especial, en l'estructura repetitiva. És per això que s'introdueix el capítol amb un repàs d'aquesta estructura.

En la majoria de programes, ens interessa indicar que una operació o més d'una s'han de repetir diverses vegades. De fet, d'alguna manera podríem dir que la potència de la programació prové precisament de la capacitat dels ordinadors de repetir una operació (o més) moltes vegades en molt poc temps. Com ja s'ha explicat al capítol 2, l'estructura repetitiva ens permet indicar que un conjunt d'operacions s'ha d'executar un nombre determinat de vegades. Al capítol anterior s'ha explicat la sintaxi de les estructures repetitives *while* i el *for*, i s'han vist exemples del seu ús en alguns problemes. En aquest capítol, s'aprofundeix l'ús de l'estructura repetitiva per resoldre problemes una mica més complexos.

Cal recordar que una part molt important de l'estructura repetitiva consisteix a definir correctament en quines condicions s'han de repetir aquestes operacions. També s'ha de triar l'estructura més apropiada per resoldre un problema determinat. L'estructura *while* es pot utilitzar en totes les situacions en què s'ha de repetir un conjunt d'operacions. Ara bé, si sabem el nombre de vegades que s'ha de repetir un conjunt d'operacions, a més del *while* podem utilitzar també l'estructura *for*.

Error freqüent: quan la condició del *while* sempre s'avalua a fals es produeix un bucle infinit.

Si el problema consisteix a tractar un conjunt de números en un interval d'enters, l'estructura *for* és més convenient, ja que permet expressar fàcilment la repetició d'accions en un interval. Un exemple d'aquest tipus de programa és el programa 13 del capítol anterior, que escriu els enters positius parells menors de 20, o una variació que es presenta a continuació, que suma els enters positius múltiples de 2 i de 3 i menors d'1000000:

Programa 30

Programa que suma els enters positius múltiples de 2 *i de* 3 *i menors d'*1000000

```
// Programa que suma tots els enters positius múltiples de 2 i de 3 menors d'1000000
#include <iostream>
using namespace std;
int main(void) {
```

```
long int num, suma;
  suma = 0;                                          // inicialització de la variable suma
  for (num = 6; num<= 1000000; num = num+1) {
     if  ((num%2 == 0) && (num%3 == 0)){             // si múltiple de 2 i 3
       suma = suma + num;                            // a suma s'hi afegeix num
     }
   }
  cout<< "La suma dels enters multiples de 2 i 3 es :  "<< suma<<endl;
system("pause");
}
```

L'estructura *for* no és la més adequada en totes les situacions. Per exemple, si es canviés una mica l'enunciat anterior i es demanés que el programa sumés els números múltiples de 2 i de 3 dels enters introduïts pel teclat, essent variable la quantitat dels enters introduïts i el valor -1 indiqués la fi dels números introduïts, el programa resultant seria bastant semblant, però s'utilitzaria l'estructura repetitiva *while*. A continuació, es detalla la resolució d'aquest nou enunciat.

Programa 31

Programa que suma els enters múltiples de 2 *i de* 3 *d'una llista acabada en* -1 *introduïda pel teclat*

```
// Programa que suma els enters positius múltiples de 2 i de 3 d'una llista acabada en -1, introduïda per teclat

#include <iostream>
using namespace std;
int main(void) {
  int num, suma;
  suma = 0;                                          // inicialització de les variables suma
  cout << "Introduiu un enter, -1 per acabar "<< endl;
  cin >>num;                                         // es llegeix el primer enter i es guarda a num
  while ( num != -1) {                               // mentre no darrer enter
     if  ((num%2 == 0)&& (num%3 == 0)){              // si l'enter és múltiple de 2 i 3
       suma = suma + num;                            // s'acumula a la variable suma
     }
      cout << "Introduiu un enter, -1 per acabar "<< endl;
      cin >>num;                                     // es llegeix el següent enter i es guarda a num
   }
  cout << "El resultat de sumar els enters multiples de 2 i 3 es: "<< suma
<< endl;
system("pause");
}
```

- La potència de la programació prové de la capacitat dels ordinadors de repetir un conjunt d'accions moltes vegades en poc temps; per tant, és necessari aprendre a utilitzar correctament les estructures repetitives.
- És important saber quina és l'estructura repetitiva més adequada en cada situació:
  - L'estructura *for* és especialment indicada per repetir un conjunt d'accions un nombre determinat de vegades.
  - L'estructura *while* es pot utilitzar en totes les situacions.

3.2 Definició de seqüències

En aquest capítol, s'analitzen amb detall programes en els quals s'han de repetir determinades accions sobre un conjunt d'elements. Per facilitar la resolució sistemàtica d'aquests problemes, es considera el conjunt d'elements que s'han de tractar com una seqüència, és a dir, una successió de valors d'un tipus determinat. Exemples de problemes de seqüències són els següents:

- **El càlcul de l'import total del carretó d'anar a comprar.** El codi de barres de cada un dels productes escollits s'ha de llegir un per un (seqüencialment) amb un lector per tal d'identificar el producte i afegir-hi el preu a l'import total.
- **El control dels vehicles que passen per un peatge d'autopista en un interval de temps determinat.** Els vehicles passen de forma seqüencial pel peatge, és a dir, cada cotxe ha d'esperar que el de davant hagi passat el peatge per pagar i passar a continuació. Si suposem que un dispositiu situat al peatge cada vegada que passa un cotxe envia a un ordinador connectat l'hora exacta, és fàcil fer un programa que compti el nombre de vehicles que hi ha passat en un interval de temps concret.
- **El control dels vehicles que hi ha en un aparcament amb barreres en una hora determinada.** Com en el cas anterior, si un dispositiu situat a la barrera envia informació a un ordinador sobre l'hora a la qual els cotxes entren i surten, aquesta informació es pot tractar com una seqüència. Fer un programa que tracta les dades generades pel dispositiu per tal de calcular el nombre de cotxes que hi ha a l'aparcament en un moment donat és fàcil: s'utilitza una variable entera, que s'incrementa cada vegada que hi entra un cotxe i es decrementa quan en surt.
- **Càlculs estadístics** (màxims, mínims, mitjanes...) sobre una llista d'elements d'una població concreta (per exemple, les notes d'una assignatura).

Característiques de les seqüències Per tractar una seqüència, sempre se n'han de determinar les tres característiques bàsiques que la defineixen i que són: el *primer element*, l'*element següent* i la *condició d'acabament*.

- **Primer element.** Tota seqüència té un primer element i s'hi ha d'accedir per tal de poder accedir a la resta.

  Per exemple, en el problema del carretó d'anar a comprar, sempre es passa un primer element pel lector de codi de barres. En el problema del control dels cotxes que passen per un peatge en un interval de temps determinat, sempre hi ha un primer cotxe, excepte en el cas que no passi cap cotxe, perquè llavors es tracta d'una seqüència buida.

- **L'element següent.** A partir del primer element, s'accedeix a la resta d'elements, segons una regla de successió.

  En el problema del carretó d'anar a comprar, després de llegir el codi del primer producte pel lector de codi de barres es llegeix el següent, i així successivament fins que no hi ha més productes. En el problema del control dels cotxes que passen per un peatge durant un període de temps determinat, el procés és el mateix: després del primer cotxe passa el següent i després el següent, fins que o bé no hi ha més cotxes o bé s'ha arribat a la fi del període controlat.

- **Condició d'acabament de la seqüència.** Per tractar qualsevol seqüència, s'ha de conèixer la informació que permet detectar-ne el final i que pot ser:
  - **El nombre d'elements de la seqüència**, és a dir, la longitud de la seqüència. Per exemple, l'enunciat indica que s'introdueix pel teclat una seqüència de 10 elements.
  - **El valor de l'últim element**, o la seva propietat. En la majoria dels casos, aquest últim element no forma part de la seqüència pròpiament, sinó que és un sentinella (una marca) i, per tant, no s'ha de tractar. Per exemple, en el darrer problema presentat d'aquest capítol, l'enunciat indica que l'enter que finalitza la seqüència és el -1; aquest element és el sentinella.

En el tractament d'una seqüència, sempre es pot distingir:

- L'element al qual s'està accedint (element en curs).
- Els elements anteriors (elements als quals s'ha accedit prèviament o part esquerra).
- Els elements següents (elements als quals encara no s'ha accedit o part dreta).

En programació, s'utilitza una estructura repetitiva per obtenir i tractar tots els elements de la seqüència. L'obtenció i el tractament de cada element depenen de l'especificació del problema. Si el programa ha de llegir la seqüència del teclat (o un altre canal d'entrada), l'obtenció de cada element (tant del primer com de la resta) es realitza repetint l'acció de llegir. De fet, la lectura pel teclat es fa sempre de manera seqüencial. Cada vegada que s'executa l'acció de llegir, se n'obté el valor següent introduït. No es pot llegir de nou un valor que s'ha llegit prèviament ni es pot deixar de llegir un valor i llegir els valors que hi ha a continuació.

En el cas de les seqüències generades pel programa a partir d'una definició (per exemple, una sèrie matemàtica), tant el primer element com la relació per obtenir els següents s'obtenen de la mateixa definició.

- Una seqüència es caracteritza per:
  - Primer element
  - Següent element
  - Final de seqüència
- En tota seqüència, sempre es pot distingir:
  - Element al qual s'està accedint (element en curs)
  - Elements anteriors (als quals ja s'ha accedit)
  - Elements següents (pendents d'accedir-hi)
- En programació, s'utilitza una estructura repetitiva per obtenir i tractar tots els elements de la seqüència.

3.3 Estratègies per tractar problemes de seqüències

En programació, com en tants camps, és útil seguir estratègies generals per resoldre problemes concrets. Per resoldre els problemes en els quals s'han de tractar seqüències d'elements, resulta útil seguir una estratègia de recorregut. La idea principal d'aquesta estratègia consisteix a recórrer la seqüència, obtenir (guardant en variables) i tractar un a un tots els elements de la seqüència. Aquesta estratègia es descriu a continuació.

3.3.1 Esquema de recorregut

L'estratègia de recorregut es presenta seguint els tres passos que s'han descrit al capítol anterior per a la construcció d'un programa.

1. Definició i inicialització de variables

   Es defineixen les variables necessàries per guardar un sol element de la seqüència i tractar-lo. En alguns problemes, s'han d'utilitzar, a més de les variables per guardar un element de la seqüència, les variables per guardar separadament el primer element.

2. Cos del programa. Obtenció i tractament dels elements de la seqüència

   El cos del programa consisteix en l'obtenció i el tractament de tots els elements de la seqüència. Per fer aquest pas primer, és necessari esbrinar les característiques de la seqüència, és a dir, com obtenir el primer element, l'element següent i quina és la condició d'acabament.

   Com ja s'ha comentat, s'utilitza una estructura repetitiva per obtenir i tractar tots els elements de la seqüència. Una vegada s'ha obtingut l'element següent, és a dir, s'ha guardat el seu valor a una o més variables, s'han de fer les operacions necessàries sobre aquestes variables. Quan l'element ja ha estat tractat, no és necessari i les variables on s'ha guardat s'utilitzen per guardar l'element següent.

```
TractamentInicial
ObtenirPrimerElement
while (!UltimElement){
        TractarElement
        ObtenirSegüentElement
      }
```

   La condició de fi de la seqüència és la condició d'acabament de l'estructura repetitiva. La condició d'aquesta estructura s'ha d'avaluar a *cert* mentre l'element obtingut sigui diferent de l'últim de la seqüència. L'elecció de l'estructura repetitiva que s'ha d'utilitzar depèn, principalment, d'aquesta condició d'acabament. Si la condició és el nombre d'elements de la seqüència, es pot utilitzar tant l'estructura *for* com el *while*; si la condició d'acabament és l'últim element (el sentinel·la), s'utilitza un *while*. És necessari obtenir el primer element de la seqüència abans del *while* per tal de poder avaluar correctament la condició la primera vegada que s'hi accedeix.

   El tractament de cada element pot ser més simple o complex, depenent de l'enunciat. Per exemple, en els dos primers programes d'aquest capítol, el tractament és simple i consisteix a sumar l'element a la variable *suma*. En alguns problemes, el tractament del primer element de la seqüència pot ser diferent del tractament de la resta d'elements.

3. Tractament final

   En la majoria dels programes, el resultat final s'escriu pel canal de sortida.

- És útil seguir estratègies generals per resoldre problemes concrets.
- L'estratègia general per tractar seqüències, coneguda com a estratègia de recorregut és la següent:

```
TractamentInicial
ObtenirPrimerElement
while (!UltimElement){
        TractarElement
        ObtenirSegüentElement
      }
TractamentFinal
```

3.3.2 Exemples de l'ús de l'estratègia de recorregut

A continuació, es descriu la utilització de l'estratègia de recorregut en la resolució de diferents tipus de problemes. De fet, alguns dels problemes del capítol 2 s'han resolt seguint aquesta estratègia. En els exemples descrits en aquesta secció, s'analitzen diferents situacions que es poden donar en els problemes de seqüències i quina és la millor estratègia que cal seguir en cada cas. Es tracta de programes senzills, en els quals s'ha de tractar una seqüència de números introduïda pel teclat (per calcular la suma o el màxim), una seqüència de caràcters i, finalment, una seqüència generada pel mateix programa.

Programa 32

Programa que llegeix pel teclat una seqüència de nombres reals que correspon als preus en euros de cada un dels productes comprats (se suposa que prèviament amb un lector de codi de barres s'ha identificat el producte i, per tant, el preu) i escriu pel canal de sortida la suma resultant. La seqüència introduïda acaba en el real $-1{,}0$, és a dir, el número $-1{,}0$ indica que no hi ha més productes.

Un exemple de programa que indirectament s'utilitza moltes vegades en la vida quotidiana: el càlcul de l'import total d'un conjunt de productes. Per calcular l'import del carretó d'anar a comprar, habitualment es passen els productes escollits un darrere l'altre pel lector del codi de barres per tal d'identificar-los i afegir-hi el seu preu a l'import total. Per simplificar, en aquest exercici se suposa que s'introdueix el preu de cada producte pel teclat (més endavant en aquest capítol es presenta un programa en què s'utilitza un lector de codi de barres com a canal d'entrada).

1. **Definició i inicialització de variables**

   En aquest exemple, es necessiten dues variables de tipus real:

   - *preu*, per guardar el preu de cada un dels productes que es van passant pel lector
   - *suma*, per guardar la suma dels preus dels productes

   A la variable *suma*, se li assigna inicialment el valor $0{,}0$, ja que és el neutre de la suma.

2. **Obtenció i tractament dels elements de la seqüència**

   - Característiques de la seqüència
     - Obtenció d'elements: els elements (tant el primer com la resta) s'obtenen utilitzant l'operació de llegir, ja que la seqüència s'introdueix per teclat.
     - Darrer element: $-1{,}0$.
   - Operacions sobre els elements: s'afegeix el preu de cada element comprat a la variable *suma*.

   És important entendre que una sola variable, *preu*, és suficient per guardar i sumar l'import de cada un dels productes. El primer element es llegeix i guarda a la variable *preu* i s'afegeix a la variable *suma*. Una vegada s'ha afegit el valor de *preu* a *suma*, és a dir, l'element ha estat tractat, ja no és necessari i, per tant, s'aprofita la mateixa variable *preu* per llegir i tractar els números següents de la seqüència.

   Seguint l'esquema de recorregut, s'utilitza una estructura repetitiva *while* per indicar que el procés de tractar un element i obtenir-ne el següent s'ha de repetir fins a obtenir el darrer element de la seqüència, en aquest cas el -1. Així doncs, les accions que s'han de repetir són:

```
suma = suma + preu;
cout <<endl<<"Entra el preu del seguent producte";
cin >> preu;
```

   La condició del *while* s'ha d'avaluar a *cert* mentre l'últim l'element llegit pel teclat sigui diferent de -1. Per això, a la condició del *while* ha d'aparèixer la variable *preu*, on es guarda l'últim element llegit. La primera vegada que s'avalua aquesta condició la variable ja ha de tenir un valor; per tant, es llegeix el primer element de la seqüència abans.

3. **Tractament final**

   El tractament final d'aquest programa consisteix a escriure el total d'euros a pagar.

El programa resultant és el següent:

```
/* Programa que calcula el preu del carretó d'anar a comprar, sumant la llista de preus
introduïda pel teclat i acabada en -1 */

#include <iostream>
using namespace std;
int main(void) {
  double preu, suma;
  suma = 0.0;
  cout <<"Introduiu el preu del primer producte;  per acabar, escriviu -1.0: ";
  cin >> preu;                                  // obtenir primer element
  while (preu != -1.0) {                        // mentre no últim element
     suma = suma + preu;
     cout <<endl<<"Introduiu el preu del producte seguent;
     cin >> preu;                               // obtenir l'element següent
   }
cout<<"El total a pagar es: "<< suma<<endl;
system("pause");
}
```

Perquè la interacció amb l'usuari sigui més fàcil, abans de llegir cada element el programa escriu per pantalla un missatge que li demana que hi entri un nou real. L'usuari també hi podria entrar la seqüència de reals separada per blancs, *return* en acabar i el mateix programa funcionaria perfectament.

- El problema del carretó d'anar a comprar és un exemple de problema que es pot resoldre seguint l'estratègia de recorregut.
- Per obtenir (i tractar) tota una seqüència, només són necessàries les variables per obtenir un sol element, ja que les variables per obtenir (i tractar) un element es reutilitzen per la resta d'elements.

Programa 33

Programa que escriu el màxim d'una seqüència donada de nombres reals positius acabada en -1

1. **Definició i inicialització de variables**

   S'utilitzen dues variables reals:

   - *num*, per guardar els números de la seqüència
   - *maxim*, per guardar el més gran dels elements tractats; s'inicialitza a $-1{,}0$, ja que la seqüència és de nombres reals positius

   La variable *maxim* també es pot inicialitzar amb el primer element de la seqüència. Si la seqüència d'entrada fos de reals positius i negatius *maxim* hauria de tenir com a valor inicial o bé el nombre real més petit possible o bé el primer element.

2. **Obtenció i tractament dels elements de la seqüència**

   - Característiques de la seqüència

     - Obtenció d'elements: s'utilitza l'operació de llegir
     - Darrer element: 0,0

   - Operacions sobre els elements:

     Es compara cada element que s'obté amb el valor guardat a la variable ***maxim*** i, en cas que l'element sigui més gran, es guarda a la variable ***maxim***. A la variable ***maxim***, hi ha en tot moment l'element més gran dels que s'han tractat.

3. **Tractament final**

   En acabar el tractament de tots els elements de la seqüència, a la variable ***maxim*** hi ha l'element més gran. En el cas concret que la seqüència sigui buida, és a dir, l'únic element sigui el valor que indica la fi de la seqüència, la variable ***maxim*** conté aquest valor. Per esbrinar si és aquest el cas i escriure el missatge final pertinent per pantalla, s'utilitza una estructura condicional.

El programa resultant és el següent:

```
// Programa que escriu el màxim d'una seqüència de nombres reals positius acabada en 0.0

#include <iostream>
using namespace std;
int main(void) {
  double num, maxim = -1;
  cout <<"Introduiu el seguent real positiu;  per acabar, escriviu 0.0: ";
  cin >> num;
  while (num != 0.0) {
    if (num > maxim) maxim = num;
    cout <<"Introduiu el seguent real positiu;  per acabar, escriviu 0.0: ";
    cin >> num;
  }
  if (maxim != -1)                    // es comprova si la seqüència és buida
     cout << "El numero mes gran es  "<< maxim <<endl;
   else
    cout <<"La sequencia es buida "<< endl;

system("pause");
}
```

- En la majoria de problemes, s'ha de considerar la possibilitat que la seqüència d'entrada sigui la seqüència buida
- Per trobar el màxim (o el mínim) d'una llista de números, s'ha d'utilitzar una variable que representi, en cada moment de l'execució del programa, l'element més gran (o més petit) dels que ja han estat tractats

Programa 34

Programa que compta les vegades que apareix la lletra 'z' en minúscula o en majúscula en una frase acabada en '.'

La resolució més apropiada del problema consisteix a considerar la frase com una seqüència de caràcters i seguir l'estratègia de recorregut.

> Els valors del tipus caràcter es representen entre comes simples (' ').

1. **Definició i inicialització de variables**

   Les variables necessàries són:

   - *c*, de tipus caràcter, per representar cada un dels caràcters que hi ha a la frase.
   - *comptador*, de tipus enter, per comptar el nombre de vegades que apareixen els caràcters 'z' o 'Z', s'inicialitza a 0.

2. **Obtenció i tractament dels elements de la seqüència**

   - Característiques de la seqüència
     - Obtenció d'elements: s'utilitza l'operació de llegir per guardar un a un els caràcters de la frase. En C++, els caràcters es poden llegir com la resta de tipus elementals, escrivint `cin >> c;` on *c* és una variable de tipus caràcter prèviament definida.

       Cal tenir present que llegint els caràcters d'aquesta manera es perd informació sobre els blancs i els salts de línia; en el cas concret que sigui necessari tractar aquests caràcters, s'ha d'utilitzar una de les diverses operacions disponibles per llegir caràcters que retornen aquests caràcters separadors (*getc*, *getchar*, ...).
     - Darrer element: el caràcter punt, que es representa com '.' .
   - Operacions sobre els elements

     Comprovar si es tracta del caràcter 'z' (en minúscula o majúscula) i, en cas afirmatiu, incrementar la variable *comptador*.

3. **Tractament final**

   En finalitzar l'obtenció i el tractament dels elements de la seqüència, la variable *comptador* conté el nombre de vegades que apareix la lletra `'z'` i s'escriu per pantalla.

> A la memòria de l'ordinador, les lletres minúscules i majúscules es representen amb enters diferents, establerts en la codificació ASCII.

El programa resultant és el següent:

```
/* Programa que compta el nombre de vegades que apareix la lletra 'z' en minúscula
   o en majúscula en una frase acabada en punt, introduïda pel teclat */

#include <iostream>
using namespace std;
int main(void) {
```

```
  int comptador= 0;
  char c;
  cout <<  "Introduiu una frase acabada en punt"<<endl;
  cin >> c;
  while (c!= '.') {
   if (c == 'z'||c== 'Z') comptador++;
   cin >> c;
  }
   cout << "El caracter 'z' apareix "<< comptador << "vegades."<<endl;
system("pause");
}
```

- Una frase es pot tractar com una seqüència de caràcters acabada amb el caràcter '.' o com una seqüència de paraules.
- Un caràcter es pot llegir amb l'operació *cin* >> *c*, on *c* és de tipus caràcter.

Programa 35

Programa que calcula les possibles pèrdues d'aplicar una rebaixa a 10 *articles d'una botiga de roba en considerar que es vendran totes les peces. El preu de cada article s'ha de rebaixar segons el nombre d'unitats existents: un* 50 % *si hi ha* 10 *unitats o més, un* 20 % *si hi ha entre* 3 *i* 9*, i si n'hi ha menys no es rebaixen. El programa ha de llegir, per cada article, el nom, el preu i les unitats restants, i n'ha d'escriure les possibles pèrdues i també el nombre d'articles rebaixats i el percentatge aplicat.*

Als exemples anteriors, s'han vist seqüències en les quals cada element estava representat per un valor (un enter, un real o un caràcter), però hi ha molts problemes en què cada element de la seqüència pot estar representat per més d'un valor, com és el cas d'aquest programa, que calcula el cost d'aplicar una possible rebaixa als productes d'una botiga de roba.

Una paraula es pot tractar com un element de tipus cadena o com una seqüència de caràcters acabada amb el caràcter blanc (' ').

1. **Definició i inicialització de variables**

   Les variables necessàries per obtenir els elements de la seqüència (els articles de roba) són tres:

   - *nom*, de tipus cadena (*string*), per representar el nom
   - *preu*, de tipus real, per representar el seu preu
   - *unitat*, de tipus enter, per representar el nombre d'unitats

   Les variables necessàries per realitzar el tractament demanat són:

   - *perdues*, de tipus real, per representar les possibles pèrdues
   - *reb50*, de tipus enter, per representar el nombre d'articles rebaixats un 50 %
   - *reb20*, de tipus enter, per representar el nombre d'articles rebaixats un 20 %

   Aquestes tres variables tenen com a valor inicial 0.

2. **Obtenció i tractament dels elements de la seqüència**

   - Característiques de la seqüència

     - Obtenir elements: Per a cada element de la seqüència, el programa llegeix les tres dades que el descriuen: el nom, el preu i el nombre d'unitats. Cal destacar que, encara que el nom de l'article no sigui necessari per realitzar els càlculs, s'ha de llegir per tal de poder obtenir la resta de la informació.
     - Condició d'acabament: L'enunciat indica que s'introduirà informació de 10 articles; per tant, la condició de fi consisteix a haver obtingut i tractat els 10 elements. El fet de conèixer el nombre d'elements de la seqüència fa apropiat l'ús de l'estructura repetitiva *for* (tot i que també es podria utilitzar un *while*).

   - Operacions sobre els elements

     El tractament de cada article de roba consisteix a comprovar la quantitat d'unitats que se'n disposen, calcular la possible pèrdua d'aplicar-hi el descompte corresponent i incrementar el comptador de roba rebaixada.

3. **Tractament final**

   En finalitzar l'obtenció i el tractament dels elements de la seqüència, el programa escriu el total de les possibles pèrdues, el nombre de productes rebaixats i els no rebaixats.

El programa resultant és el següent:

```
/* Programa que llegeix la descripció de 10 articles de roba (nom, preu i unitats restants)
    i calcula les possibles pèrdues d'actualitzar-ne el preu si es rebaixen un 50% tots
    els articles dels que disposa de 10 unitats o més i un 20% dels que es disposa d'entre 3 i 9 unitats.
    També escriu el nombre de productes rebaixats al 50%, al 20% i els no rebaixats */

#include <iostream>
using namespace std;
int main(void) {
  string nom; double preu,perdues = 0.0; int unitats;
  int reb50=0, reb20 = 0,i;

for(i=0; i<10; i++){
  cout << "Entra el nom, el preu i les unitats del producte seguent"<<endl;
  cin >> nom >>preu >> unitats;
  if ( unitats >= 10)                    // si hi ha més de 10 unitats es rebaixen al 50%
      { perdues = perdues + preu*0.5*unitats; reb50++;}
          else if (unitats >= 3)// si hi ha entre 3 i 9 unitats es rebaixen al 20%
   { perdues = perdues + preu*0.2*unitats; reb20++;}
  }     // fi del for
  cout << "Possibles perdues: "<< perdues << "euros"<<endl;
  cout << "El nombre de productes rebaixats al 50%  es "<< reb50 <<endl;
  cout << "El nombre de productes rebaixats al 20%  es "<< reb20 <<endl;
  cout << "El nombre de productes no rebaixats es "<< 10 - reb50-reb20 <<endl;
  system("pause");
}
```

- Els elements d'una seqüència poden estar representats per més d'un valor de diferent tipus i s'ha de guardar cada un d'aquests valors en variables diferents.
- En C++, una paraula es pot guardar en una variable de tipus cadena *(string)*.

Programa 36

Programa que calcula el valor aproximat del cosinus de x, amb una precisió determinada

El valor aproximat del cosinus d'una x donada es pot calcular a partir del desenvolupament de la sèrie de Taylor per a la funció *cosinus(x)*. És un exemple de problema que treballa amb una sèrie matemàtica i, com ja s'ha indicat, les sèries matemàtiques es poden considerar seqüències. La condició d'acabament de la seqüència ve donada per la precisió exigida per l'usuari. Per a realitzar una solució més completa, el programa controla que la precisió estigui entre 0 i 1, i també escriu el nombre de termes de la sèrie que són necessaris per aconseguir la precisió demanada.

Cal recordar que el desenvolupament de Taylor de la funció *cosinus(x)* és el següent:

$$cosinus\,(x) = 1 - (x^2/2!) + (x^4/4!) - (x^6/6!) + \cdots + (x^n/n!) \tag{3.1}$$

on n comença valent 0 i s'incrementa en 2 unitats en cada terme.

1. **Definició i inicialització de variables**

   El programa utilitza diverses variables reals i enteres per fer els càlculs.

   Les variables reals són les següents:

   - x representa el valor del qual el programa calcula el cosinus, introduït per l'usuari,
   - *precisio*, la precisió introduïda per l'usuari (entre 0 i 1),
   - *cosaprox*, el cosinus calculat pel programa, inicialitzada a $0{,}0$,
   - *terme*, el valor de l'últim terme tractat, inicialitzada amb el valor del primer element, el $1{,}0$.
   - *exp*, exponencial de x (utilitzada en el càlcul del terme següent), inicialitzada a 1.

   Les variables enteres utilitzades són:

   - *nterme*, el nombre de termes de la sèrie ja tractats s'inicialitza a 0,
   - n, utilitzat per calcular cada terme, correspon al doble del nombre de termes de la sèrie ja tractats,
   - *fact*, el factorial de n (que s'ha utilitzat en el càlcul del terme següent), s'inicialitza a 1.

2. **Obtenció i tractament dels elements de la seqüència**

   - Característiques de la seqüència

     Per definició, un terme de la sèrie és: $x^n/n!$, on n és igual al nombre de termes anteriors multiplicat per 2 i comença valent 0. S'utilitza la variable *term* per guardar un terme, *exp* per guardar l'exponencial de x (el numerador) i *fact* per guardar el factorial de n (el denominador):

     ```
     terme = exp/fact;
     ```

- Primer element. Quan n és 0 *terme* és 1, ja que x^0 és 1 i 0! per definició és 1.
- Element següent. El més eficient és calcular l'exponencial (el numerador) i el factorial (el denominador) del terme següent a partir del terme anterior.

  Per calcular x^n, es pot utilitzar la funció predefinida *pow*, però és més eficient aprofitar que el valor de *exp* per l'element anterior és x^{n-2} i fer:

```
exp = exp*x*x;
```

  Per calcular $n!$, on $n! = n*n-1*n-2*\cdots*1$ i guardar el resultat a *fact* es pot fer:

```
fact = 1;        // inicialització de la variable fact
for (i = 2; i<= n; i++) {
   fact = fact * i;       // producte de fact per la unitat següent
 }
```

  Però és més eficient aprofitar que el valor de *fact* per l'element anterior és $n-2!$ i fer:

```
fact= fact*(n -1)* n;
```

  El signe del nou terme depèn de si es tracta d'un nombre de terme parell (signe positiu) o senar (signe negatiu):

```
if(nterme%2 == 1)
          terme = -terme;
```

- Darrer element. Terme amb valor absolut menor a la precisió demanada. La sèrie va convergint, és a dir, el valor absolut de cada nou terme és menor al del terme anterior i sempre s'acaba obtenint un terme que en valor absolut és més petit que la precisió exigida.

- Operacions sobre els elements: sumar el nou terme a la variable que representa el cosinus aproximat.

3. Tractament final

   L'últim pas del programa consisteix a escriure el cosinus aproximat obtingut i el nombre de termes que s'han necessitat per obtenir-lo.

El programa resultant és el següent:

```
/* Programa que calcula el valor aproximat del cosinus d'una x donada amb una precisió determinada
   a partir del desenvolupament de la sèrie de Taylor per la funció cosinus */

#include <iostream>
using namespace std;
int main(void){
    double cosaprox= 0.0;          //   cosinus calculat pel programa
    double x;                      // valor del qual s'ha de calcular el cosinus
    double precisio;               //   precisió exigida per l'usuari
    double terme = 1.0;            // últim terme tractat del desenvolupament de Taylor
    double  exp = 1;                 // potència de x
    int n;                                 //   utilitzat per tractar el següent terme de la sèrie
    int nterme = 1;                //   número de termes de la sèrie tractats
    int fact = 1;                        // factorial de x
    cout<<"Introduiu el valor del que voleu calcular el cosinus"<<endl;
    cin>>x;
```

```
    cout << "Introduiu la precisio (major que 0 i menor que 1)";
    cin >> precisio;
    while (precisio<0 ||precisio>1) {     //la precisió donada està entre 0 i 1
        cout << "Introduiu precisio (major que 0 i menor que 1)";
        cin >> precisio;
       }
   while( abs(int(terme)) > precisio){     // mentre l'últim terme sigui més gran que la     precisió
       n = 2* nterme;                                          //    s'obté el següent n a partir de nterme
       exp = exp*x*x;                    // exponencial a partir de l'exponencial terme anterior
       fact= fact*(n -1)* n;             // factorial a partir del factorial del terme anterior
       terme = exp/fact;                          // següent terme de la sèrie
       if(nterme%2 == 1)             // si el número de terme és senar es canvia el signe del terme
           terme = -terme;
       cosaprox = cosaprox + terme;
       nterme++;
    } // fi del while
cout<<"Valor del cosinus aproximat: "<< cosaprox<<endl;
cout<< "Nombre de termes necessitats: "<< nterme -1<<endl;
system("pause");
}
```

- Un programa pot generar un nombre finit d'elements d'una sèrie matemàtica a partir de la seva definició, seguint l'estratègia de recorregut.
- Si es vol obtenir el resultat aproximat d'una funció matemàtica (com el cosinus o sinus) mitjançant el desenvolupament de la sèrie de Taylor corresponent, la precisió exigida determina el nombre de termes de la sèrie a tractar, és a dir, la condició de fi de l'estructura repetitiva.

3.4 Estratègies per tractar problemes de cerca

Un tipus de problema bastant freqüent consisteix a cercar un element amb determinades propietats a una seqüència. Aquest tipus d'enunciats es poden resoldre amb l'estratègia descrita de recorregut, però encara se n'obté una solució més eficient si se segueix una estratègia específica per a la cerca d'un element.

3.4.1 Esquema de cerca

En els problemes de cerca, la solució s'obté quan es troba l'element buscat; per tant, una vegada s'ha trobat l'element no és necessari tractar la resta de la seqüència. En el cas concret que l'element cercat no es trobi a la seqüència, s'han de tractar tots elements per tal de poder concloure que no hi és. La manera més eficient per resoldre aquests problemes consisteix a adaptar l'estratègia ja descrita de recorregut perquè en el moment en què es trobi l'element es pugui finalitzar el programa, sense tractar la resta de la seqüència. A continuació, es descriu amb més detall aquesta nova estratègia.

1. Definició i inicialització de variables

   A més d'utilitzar les variables necessàries per guardar un sol element de la seqüència i tractar-lo, és convenient (si bé no imprescindible) treballar amb una variable booleana, que es pot anomenar *trobat* i que representa l'estat en què es troba la cerca de l'element. Inicialment, a la variable *trobat* se li assigna *fals*, ja que encara no s'ha trobat l'element cercat, en el moment en què es troba se li assigna el valor de *cert*.

2. Cos del programa

Aquest pas consisteix a repetir el procés d'obtenir l'element següent i comprovar si és el que se cerca mentre no s'obtingui l'últim element i no s'hagi trobat. De fet, es pot veure com una adaptació del segon pas de l'esquema de recorregut, en què el procés d'obtenir i tractar els elements de la seqüència s'atura a trobar l'element cercat.

Com en el cas dels problemes de recorregut, és necessari esbrinar les característiques de la seqüència: primer element, element següent i darrer element. També s'utilitza una estructura repetitiva *while* per repetir el procés d'obtenir l'element següent i comprovar si compleix les característiques buscades. Per aturar aquest procés en el moment en què es trobi l'element s'utilitza la variable booleana *trobat*. Aquesta variable s'incorpora a la condició d'acabament del *while*, de forma que el procés de cerca s'atura o bé en arribar a la fi de la seqüència o bé quan es troba l'element cercat (el valor de la variable booleà *trobat* és *cert*).

L'esquema d'aquest segon pas és el següent:

```
TractamentInicial
ObtenirPrimerElement
while (!UltimElement && !trobat){
      if(PropietatCercada) {
           trobat = true;
      } else{
           TractarElement
           ObtenirSegüentElement
        }
   }
```

És important entendre que inicialment *!trobat* s'avalua a *true*, ja que la variable *trobat* s'ha inicialitzat a *false* (i *!false* és *true*). La primera sentència que s'ha de repetir a l'estructura *while* consisteix a comprovar si l'últim element obtingut és el cercat i, si és així, a la variable *trobat* se li assigna *true*. Quan el valor de *trobat* és *true*, la condició del *while* s'avalua a *false*, ja que *!trobat* és *false* i això implica la fi de la iteració.

La variable *trobat* a la condició del *if* és una expressió i s'avalua al valor de la variable.

3. Tractament final

En el tercer pas de la resolució del problema, es determina si s'ha trobat o no l'element cercat. S'utilitza una estructura *if*, en què la condició és la variable *trobat*: si *trobat* té valor *cert*, la condició s'avalua a *cert*, altrament s'avalua a *fals*:

```
if (trobat) { TractamentTrobat}
 else {TractamentNoTrobat}
```

- Els problemes de cerca es poden resoldre seguint l'estratègia de recorregut, però és millor seguir l'estratègia específica de cerca.
- L'estratègia de cerca és una adaptació de la de recorregut, que permet acabar l'obtenció i el tractament de la cerca en trobar l'element buscat.
- L'estratègia general per resoldre problemes de cerca és la següent:

```
bool trobat = false;
TractamentInicial
ObtenirPrimerElement
while (!UltimElement && !trobat){
      if(PropietatCercada) {
```

```
            trobat = true;
        } else{
            TractarElement
            ObtenirSegüentElement
        }
    }
    if (trobat) { TractamentTrobat}
     else {TractamentNoTrobat}
```

3.4.2 Exemples de l'ús de l'estratègia de cerca

A continuació, s'analitza com seguir aquesta estratègia per resoldre tres problemes de cerca. El primer exemple consisteix a cercar un enter en una seqüència introduïda pel teclat. En el segon programa, es busca un element que compleixi una propietat determinada, concretament ser divisor d'un número n donat. Finalment, el tercer exemple no és un problema típic de cerca, tot i que es pot resoldre aplicant la mateixa estratègia, ja que no es tracta de buscar un element concret sinó de detectar si en una seqüència donada es compleix una propietat determinada i, en aquest cas, acabar el programa.

Programa 37

Programa que, donat un nombre real i una seqüència de reals positius acabada en $-1{,}0$, determina si el nombre pertany a la seqüència.

Es tracta d'un exemple de problema on s'ha de buscar un element determinat en una seqüència; així doncs, per resoldre'l s'utilitza l'estratègia de cerca.

1. **Definició i inicialització de variables**

   Les variables utilitzades són les següents:

   - *num*, de tipus real, per guardar cada un dels elements de la seqüència,
   - *numbuscat*, de tipus real, per guardar el nombre que es vol cercar,
   - *trobat*, de tipus booleà, a la qual s'assigna el valor inicial *false*.

2. **Cerca d'un element a la seqüència**

   Inicialment, es llegeix l'element a cercar i, a continuació, la seqüència.

   - Característiques de la seqüència
     - Obtenir elements: Donat que la seqüència consisteix en una llista de reals introduïda pel canal d'entrada, s'obtenen tots els elements utilitzant l'acció de llegir.
     - Darrer element: el sentinella -1, indicat a l'enunciat.
   - Cerca

     Seguint l'esquema de cerca, cada vegada que s'obté l'element següent de la seqüència es compara amb l'element que se cerca, *numbuscat*, utilitzant una estructura condicional. En el moment en què es troba l'element, a la variable booleana *trobat* se li assigna *cert*.

   La condició d'acabament de l'estructura *while* és que, o bé l'últim element llegit és el -1 o bé el valor de la variable *trobat* és *cert*.

3. **Tractament final**

   Seguint l'estratègia de cerca, s'utilitza una estructura condicional per determinar si s'ha trobat l'element cercat o no i escriure el missatge corresponent.

```
if (trobat)
    cout <<"El nombre buscat es troba a la sequencia"<<endl;
    else
    cout<<"El nombre buscat no es troba a la sequencia"<<endl;
```

El programa resultant és el següent:

```
// Programa que cerca un número donat en una llista de reals acabada en -1

#include <iostream>
using namespace std;
int main(void) {
  double numbuscat, num;
  bool trobat = false;
  cout <<"Introduiu el nombre real a buscar ";
  cin >> numbuscat;
  cout <<endl<<"Introduiu el primer real;  per acabar, escriviu -1: ";
  cin >> num;
  while (num!= -1 && !trobat) {
    if (numbuscat == num) trobat = true;
    else {
      cout <<endl<<"Introduiu el seguent real;  per acabar, escriviu -1: ";
      cin >> num;
    }
  }
if (trobat) cout <<"El nombre buscat es troba a la sequencia"<<endl;
    else cout<<"El nombre buscat no es troba a la sequencia"<<endl;

system("pause");
}
```

> L'esquema de cerca és molt útil per detectar si un element amb determinades propietats és a la seqüència.

Programa 38

Programa que determina si un nombre enter positiu és primer.

Cal recordar que un nombre n és primer si no té altres divisors que el número 1 i el mateix n. El problema de determinar si un nombre enter és primer es pot plantejar com el problema de trobar un divisor del nombre diferent de 1 i ell mateix; per tant, es tracta d'un problema de cerca, tot i que d'entrada podria no semblar-ho.

1. **Definició i inicialització de variables**

   El programa utilitza tres variables:

   - n, de tipus enter, per guardar l'element introduït,
   - *divisor*, de tipus enter, per representar la seqüència de possibles divisors de n,
   - *trobat*, de tipus booleà, a la qual s'assigna el valor inicial *false*.

2. **Cerca d'un element a la seqüència**

Primerament, s'ha de considerar el cas especial en què el nombre introduït sigui l'enter 1, ja que per definició l'1 no és primer. S'utilitza l'estructura condicional per determinar si es tracta d'aquest cas, altrament es tracta la seqüència formada pels possibles divisors entre el 2 i el $n-1$.

- Característiques de la seqüència

  Es pot limitar la seqüència dels possibles divisors de n utilitzant la propietat matemàtica que indica que, si un nombre no té un divisor entre els enters menors o iguals a la seva arrel quadrada, és primer.

  - Primer element. El primer possible divisor:

    $$divisor = 2;$$

  - Element següent. El següent possible divisor:

    $$divisor = divisor + 1;$$

  - Darrer element. El primer enter major a l'arrel quadrada de n

- Cerca

  El primer divisor del nombre introduït és el primer element generat (guardat a *divisor*) que divideixi exactament el n, és a dir, en fer la divisió entera de n i *divisor* el mòdul és 0.

Seguint l'estratègia general de cerca, el programa utilitza una estructura repetitiva *while* per generar la seqüència d'enters fins a trobar un divisor o un enter major a l'arrel quadrada de *n* que indiqui que no hi ha més divisors possibles.

3. **Tractament final**

Per esbrinar si l'estructura repetitiva ha acabat en trobar un divisor o bé en generar l'últim divisor possible, se segueix l'esquema de cerca i es consulta la variable *trobat*. En aquest cas concret, si el valor de la variable és *cert*, vol dir que s'ha trobat un divisor i, per tant, el nombre no és primer; si el valor és *fals*, no s'han trobat divisors i, per tant, el nombre és primer.

El programa resultant és el següent:

```
// Programa que, donat un enter positiu, indica si és primer o no.

#include <iostream>
using namespace std;
int main(void) {
   bool trobat;
   int divisor, n;
   cout <<"Introduiu un enter positiu ";
   cin >> n;
   if (n == 1) {
      trobat = true;      //per definició el número 1 no és primer
   } else {
      trobat = false;
      divisor=2;
      while (!trobat && divisor*divisor <= n) {
         if (n % divisor == 0){
            trobat = true;
          } else {
```

```
            divisor++;
            }
        }
    }
  if (trobat) cout <<"El nombre no es primer"<<endl;
    else cout<<"El nombre es primer"<<endl;
system("pause");
}
```

Es podria millorar el programa si, abans de buscar tots els possibles divisors, es comprova si el nombre és parell. En cas que el nombre sigui parell, no és necessari realitzar cap altre tractament; en cas contrari, només cal buscar un possible divisor entre els nombres senars. Concretament, si el nombre és imparell, la seqüència a tractar seria:

- Primer element. El primer possible divisor: $divisor = 3$.
- Element següent. El següent possible divisor: $divisor = divisor + 2$.
- Darrer element. El primer enter major a l'arrel quadrada de n.

> Hi ha problemes que es poden plantejar com a problemes de cerca, tot i que l'enunciat no indica explícitament que s'ha de cercar un element.

Programa 39

Programa que actualitza la quantitat de llibres disponibles de quatre tipus diferents. Per a cada tipus de llibre, es disposa del codi de barres, el preu assignat i la quantitat en stoc*. L'operari introdueix (mitjançant el teclat o a través del lector) el codi de barres del llibre i, per finalitzar, s'introdueix el* 0*. El programa s'interromp si la quantitat de llibres venuts és igual a la quantitat de llibres existents (la suma de tots els tipus). El programa escriu un missatge d'avís per reposar l'*stoc *si el nombre de llibres existents d'un tipus és inferior a* 30% *de la quantitat màxima. Una vegada acabada la venda, escriu els següents resultats: número total de llibres venuts, número de llibres de cada tipus existents en* stoc*, un missatge d'avís per reposar l'*stoc *si el nombre de llibres existents a l'*stock *es inferior al* 30% *de la quantitat màxima (per cada tipus de llibre).*

Aquest problema també es pot resoldre aplicant l'estratègia de cerca, ja que el que s'ha de determinar és si l'últim element obtingut determina el final del programa. Cal entendre que el mateix programa pot servir quan la seqüència s'introdueix pel teclat o pel codi de barres, ja que el programa rep la mateixa informació, independentment del dispositiu d'entrada. A continuació, es presenten algunes indicacions sobre l'ús del lector de codi de barres.

1. Connectar la sortida del lector de codi de barres a l'entrada USB del PC. El lector (escànner) del codi de barres no necessita alimentació externa ja que s'alimenta a través del mateix port USB.
2. Esperar que el sistema operatiu reconegui el dispositiu correctament.
3. Quan es passa un codi de barres pel lector, la funció de l'escànner és llegir el codi de barres, traduir-lo en una dada i enviar-lo al PC com si s'hagués escrit pel teclat el codi de barres. Es tracta el codi de barres com una cadena (*string*); per tant, en el programa realitzat en C++, es pot escriure:

```
string cb;
cout << "Introduiu un codi de barres "<< endl;
```

El programa resultant seria

```
// Programa que actualitza els estocs de llibres

#include<iostream>
#include<string>
using namespace std;
const int STOCK1 = 15;
const int STOCK2= 12;
const int STOCK3 = 5;
const int STOCK4 = 10;
int main()
{
    int cont1 = 0, cont2 = 0, cont3 = 0, cont4 = 0;
    bool trobat = false, hihastock1 = true, hihastock2 = true, hihastock3 = true,
hihastock4 = true;
    string cb;
    cout << "Introduiu un codi de barres d'un llibre disponible,
ja sigui pel lector de codis de barres o per teclat "<< endl;
    cin>>cb;
    while(cb != "0"&& !trobat)
       if(cb == "CB1")        {
              if(hihastock1)
              {
                  cont1++;
                  if(cont1 == STOCK1
                      {hihastock1 = false;                  }
              }
              else
                  cout << "Aquest producte esta exhaurit!"<< endl;
       }
       else if(cb == "CB2")
       {
              if(hihastock2)
              {
                  cont2++;
                  if(cont2 == STOCK2)
                  {
                      hihastock2 = false;                  }
              }
              else
                  cout << "Aquest producte esta exhaurit!"<< endl;
       }
       else if(cb == "CB3")
       {
              if(hihastock3)
              {
                  cont3++;
                  if(cont3 == STOCK3)
                  {
                      hihastock3 = false;                  }
              }
              else
```

```
                cout << "Aquest producte esta exhaurit!"<< endl;
        }
        else if(cb == "CB4")
        {
              if(hihastock4)
              {
                  cont4++;
                  if(cont4 == STOCK4)
                  {
                      hihastock4 = false;                  }
              }
              else
                  cout << "Aquest producte està exhaurit!"<< endl;
        }
        else
        {
            cout << "Codi incorrecte! Introduiu un dels 4 valors correctes!"<<
endl;
        }

        // Mentre hi hagi estoc d'algun producte, es poden continuar venent els productes existents

        if ( !hihastock1 &&  !hihastock2  && !hihastock3 && !hihastock4) {
         trobat = true;
        } else {
         cout << "Introduiu un codi de barres d'un llibre disponible, ja sigui
pel lector de codis de barres o per teclat "<< endl;
          cin>>cb;
        }
    }
    if(trobat)
        cout << "S'han acabat tots els estocs!"<< endl;
    cout << "S'han venut "<< (cont1+cont2+cont3+cont4) << "llibre(s)."<< endl;
    cout << "Els estocs actuals són els següents: "<< endl;
    cout << "Llibre CB1 "<< STOCK1 - cont1 << endl;
    cout << "Llibre CB2 "<< STOCK2 - cont2 << endl;
    cout << "Llibre CB3 "<< STOCK3 - cont3 << endl;
    cout << "Llibre CB4 "<< STOCK4 - cont4 << endl;
    if((STOCK1 - cont1) <= STOCK1*0.3)
        cout << "ALERTA! Cal comprar mes llibres CB1!"<< endl;
    if((STOCK2 - cont2) <= STOCK2*0.3)
        cout << "ALERTA! Cal comprar mes llibres CB2!"<< endl;
    if((STOCK3 - cont3) <= STOCK3*0.3)
        cout << "ALERTA! Cal comprar mes llibres CB3!"<< endl;
    if((STOCK4 - cont4) <= STOCK4*0.3)
        cout << "ALERTA! Cal comprar mes llibres CB4!"<< endl;
    system("pause");
}
```

3.5 Exemples dirigits

Completeu els programes següents:

1. Programa que escrigui els enters entre 0 i 300 múltiples de 3 i que tinguin com a última xifra el 5.

[Solució]

Es tracta d'un problema de recorregut. L'enunciat indica l'interval de nombres que s'han de tractar (el primer i l'últim element de la seqüència); per tant, es pot utilitzar l'estructura repetitiva *for*.

Aquesta estructura *for* es podria escriure de diverses formes; per exemple, de manera que directament es generi la seqüència de nombres múltiples de 3:

```
for (k = 3; k <= 300; k= k+3)
```

Utilitzant aquesta estructura repetitiva (o una altra que dissenyeu), completeu el programa seguint l'estratègia de recorregut, on el tractament de cada element consisteix a comprovar si el número acaba en 5 i, en cas afirmatiu, escriure'l per pantalla.

2. Feu un programa que avaluï un polinomi en un punt. Pel teclat arribarà primer el valor que es vol avaluar, el polinomi i, a continuació, la seqüència formada pel grau i el coeficient de tots els termes del polinomi. La seqüència acabarà en $-1-1{,}0$.

[Solució]

Es tracta d'un problema de recorregut; per tant, podem seguir l'estratègia general per a problemes d'aquest tipus. En aquest exemple, com que no sabem el nombre d'elements que s'han de tractar, utilitzarem l'estructura repetitiva *while*, en què la condició serà:

```
while (grau != -1 || coeficient != -1)
```

suposant *grau* i *coeficient* variables definides prèviament i que tindran com a valors, respectivament, el grau i el coeficient de l'últim terme tractat.

El tractament de cada terme consistirà a elevar el valor introduït al grau corresponent, multiplicar-lo pel coeficient i sumar-lo a la variable que representa la suma de tots els termes.

3. Es demana implementar un programa que calculi la longitud d'un polígon. Pel teclat s'introduirà la seqüència de punts que corresponen als vèrtexs del polígon, on cada punt estarà format per la coordenada x i la coordenada y. La seqüència de punts acabarà amb la repetició del primer punt introduït.

[Solució]

Es tracta també d'un problema de recorregut. Com que no es coneix el nombre de vèrtexs del polígon, s'utilitza l'estructura repetitiva *while*, en què la condició és:

```
while (x != x0 || y != y0)
```

on $x0$ i $y0$ són les coordenades del primer punt, i x i y les de l'últim punt llegit.

Per calcular l'arrel quadrada d'un número, es pot utilitzar la funció predefinida en C++ *sqrt(x)*.

4. Es vol dissenyar un programa que, a partir de la qualificació d'una pel·lícula i l'edat de l'espectador, digui si aquest hi pot accedir o no. La codificació de les pel·lícules es fa mitjançant una lletra, tal com es mostra a continuació:

   - T - Tots els públics.
   - A - Majors de 13 anys.
   - B - Majors de 16 anys.
   - C - Majors de 18 anys.

   El programa demanarà la lletra que determina la qualificació de la pel·lícula i l'edat de l'espectador i, a continuació, mostrarà per pantalla si aquest pot passar a la sala o no. El programa ha de controlar que la qualificació de la pel·lícula sigui T, A, B o C. Qualsevol altra entrada no serà vàlida i s'informarà l'usuari de l'error. El programa haurà de demanar repetidament dades per comprovar fins que s'introdueixi la lletra X.

   [Solució]

   El conjunt de dades que el programa ha de comprovar es pot veure com una seqüència acabada a la lletra X, en què cada element està format pel caràcter que descriu la pel·lícula i l'edat de l'espectador. S'han de tractar tots els elements de la seqüència; per tant, es pot seguir l'estratègia de recorregut. En aquest cas, el tractament de cada element consistirà a comprovar si l'edat de l'espectador es correspon amb el tipus de pel·lícula i s'implementarà amb una estructura condicional.

5. Es vol fer un programa per confeccionar el rebut d'electricitat d'un conjunt d'usuaris. El càlcul per a cada usuari es farà a partir dels kW consumits que s'han de calcular un cop introduïdes pel teclat la lectura anterior i l'actual, llegides des del comptador. S'indicarà que no hi ha informació relativa a més usuaris entrant una lectura negativa. En cas que el consum sigui un valor negatiu, s'han d'informar l'usuari i tornar a demanar les dades fins que sigui un valor positiu. Per calcular la quantitat total a pagar, s'ha de saber que hi ha 4 euros de quota fixa i que el consum es paga per trams:

   - Tram 1: els primers 300 kW es paguen a 0,5 euro/kW.
   - Tram 2: els kW entre 301 i 800 es paguen a 0,75 euro/KW.
   - Tram 3: els kW a partir de 801 es paguen a 1,5 euro/kW.

   Sabent que, a més a més, es paga un 16% d'IVA, feu un programa que mostri per pantalla la quantitat total a pagar i el que correspon a cada tram.

   [Solució]

   Es tracta d'un altre problema de seqüències, en el qual també s'han de tractar tots els elements. En aquest exemple, cada element de la seqüència ve donat per dos enters, que corresponen a la lectura anterior i actual del comptador d'electricitat d'un usuari. El tractament de l'element consisteix a calcular la quantitat a pagar per cada usuari i, per a fer-ho, s'ha de comprovar quina part del consum de cada usuari correspon al primer tram, al segon o al tercer.

   En aquest programa, es podria utilitzar l'estructura repetitiva *for*? Per què?

6. Programa que determina si el caràcter 'a' apareix més de 10 vegades en una frase acabada en punt, introduïda pel teclat.

   [Solució]

   Es tracta d'un problema de cerca, on la propietat cercada és que el caràcter 'a' apareix més de 10 vegades. Només cal que completeu el programa, que, seguint l'estratègia de cerca, resol el problema.

```
#include <iostream>
using namespace std;
int main(void) {
  int comptador= 0; bool trobat = false;
  char c;
  cin >> c;
  while (c!= '.'&& !trobat) {
   if (??) { // ** falta completar la condició de trobat
     trobat = true;
   } else { // ** falta completar el tractament de no trobat
   cin >> c;
   }
  } // fi del while
  if (trobat)
     cout << "El caracter 'a' apareix mes de "<<10 <<"vegades."<<<endl;
   else cout << "El caracter 'a' NO apareix mes de "<<10 <<"vegades. "<<<endl;
}
```

7. El programa següent determina si hi ha més d'un alumne amb 10 en una seqüència introduïda pel teclat en què apareixen els dos cognoms i la nota de l'examen de cada estudiant. La seqüència acaba en *fi fi 0*. El programa escriu si hi ha un sol estudiant amb 10 i, en cas afirmatiu, els seus dos cognoms. Es demana fer-hi els canvis pertinents per tal que el programa funcioni en cas que el darrer element introduït sigui només la paraula *fi* (enlloc de *fi fi 0*).

[Solució]

També es tracta d'un problema de cerca, i el que s'ha de fer és canviar l'obtenció de l'element següent de la seqüència en el programa següent:

```
//programa que busca si hi ha més d'un alumne amb la nota 10
#include <iostream>
using namespace std;
int main(void) {
  double  nota;
  int comptador = 0;
  string cognom1, cognom2,auxcognom1, auxcognom2;
  bool trobat = false;

  cout <<"Introduiu el cognom i la nota del primer estudiant; per acabar, fi fi 0:";
  cin >> cognom1>> cognom2>>nota;
  while (cognom1!= "fi"&& !trobat) {
    if (nota == 10 && comptador == 1) {     // si la nota és 10 i comptador és 1
          trobat = true; }
    else {
        if (nota == 10){                    // guarda el cognom del primer alumne amb 10
          comptador = 1;
          auxcognom1 = cognom1; auxcognom2 = cognom2;
          }
        cout <<"Introduiu el cognom i la nota del seguent; per acabar, fi fi 0:";
        cin>> cognom1>>cognom2>>nota;
     }
```

```
    } // fi del while
if (trobat) cout <<"Hi ha mes d'un alumne amb 10"<<endl;
     else
       if (comptador == 1)
            cout<<"Hi ha un unic alumne amb un 10 "<<auxcognom1 <<" "<< auxcognom2 ;
          else
            cout<<"NO hi ha cap alumne amb 10"<<endl;
system("pause");
}
```

3.6 Problemes avançats

Molts problemes de seqüències tenen una dificultat més gran que els programes que s'han descrit en les seccions anteriors. Hi ha problemes en què el procés d'obtenir l'element següent de la seqüència presenta alguna dificultat, ja que per tractar cada element és necessari considerar també alguns elements anteriors i/o posteriors; es pot considerar que aquests problemes han de tractar una finestra d'elements, en què la longitud de la finestra depèn de l'enunciat. Hi ha altres problemes en què el tractament de cada element de la seqüència pot tenir alguna dificultat, per exemple, quan implica treballar amb una (sub)seqüència i, per tant, el programa s'ha de plantejar com una seqüència de seqüències. En aquesta part del capítol, s'estudien alguns exemples d'aquests dos tipus problemes.

3.6.1 Problemes de finestres

Problemes com el d'esbrinar si una seqüència és creixent, el de trobar el màxim local en una seqüència de nombres o el de comptar la repetició d'un grup de caràcters consecutius en una frase són exemples de problemes de *finestra*, en què no es pot tractar cada un dels elements de la seqüència de forma aïllada, sinó que s'han de considerar tots els elements de la finestra. Cada vegada que es tracta un nou element de la seqüència, la finestra es desplaça una posició. Una finestra pot estar formada per l'element que es vol tractar i també per elements anteriors i posteriors. En la resolució d'aquests problemes, és molt important representar adequadament els elements de la finestra: s'ha de guardar en variables separades cada un dels elements que es vol considerar (l'actual, l'anterior, el posterior ...) i s'actualitzen cada vegada que s'obté un nou element.

A continuació, s'analitzen dos exemples de problemes en què es treballa amb finestres.

Programa 40

Programa que escriu els n primers números de la sèrie de Fibonacci.

La sèrie de Fibonacci és definida com:

$$f_1 = 1$$
$$f_2 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

Això és, els primers elements de la sèrie són:

$$1\ 1\ 2\ 3\ 5\ 8\cdots$$

El programa ha de tractar sempre els n termes de la sèrie; per tant, es tracta d'un problema de recorregut i se segueix l'estratègia que hem vist per tractar aquest tipus de problemes.

1. **Definició i inicialització de variables**

   La definició de la sèrie ens indica que tot element (excepte el primer i el segon, que tenen valor 1) s'obté sumant el predecessor i l'anterior al predecessor. Així doncs, per obtenir cada element de la sèrie, s'han de sumar els dos enters immediatament anteriors. Per tant, el programa ha de treballar amb una finestra de tres enters, i per a això es defineixen tres variables enteres, que s'utilitzen per generar els elements de la sèrie. També s'utilitza una variable entera per guardar el valor introduït pel teclat que representa el nombre de termes de la sèrie que s'han d'escriure per pantalla. Les variables utilitzades són les següents:

   - f, per guardar l'element que s'està tractant,
   - *fanterior*, per guardar l'element anterior a f,
   - *fanterior2*, per guardar l'element anterior a *fanterior*,
   - n, nombre de termes que s'han de tractar.

   Inicialment, el valor de *fanterior2* i *fanterior* és 1, perquè són els dos primers elements de la sèrie i la definició així ho indica.

2. **Obtenció i tractament dels elements de la seqüència**

   Per fer el programa més complet, es pot incloure la comprovació que el valor introduït pel teclat sigui major que 1. Si és així, es tracta la seqüència:

   - Característiques de la seqüència
     - Primer element. Per definició, els 2 primers termes són 1.
     - Element següent. Per definició, s'obté l'element següent sumant els dos anteriors. Abans, però, s'ha de desplaçar la finestra, és a dir, actualitzar correctament les variables que representen els dos elements immediatament anteriors:
       - L'element anterior al tractat s'assigna a la variable que representa l'anterior a l'anterior:

         $$fanterior2 = fanterior;$$

       - L'element ja tractat a la variable que representa l'anterior al nou terme:

         $$fanterior = f;$$

       Actualitzades les variables, s'obté el terme següent sumant les variables *fanterior2* i *fanterior*:

       $$f = fanterior + fanterior2;$$

     - Condició d'acabament

       El nombre de termes que s'ha de generar i tractar s'introdueix pel teclat; per tant, es pot utilitzar tant l'estructura repetitiva ***for*** com la ***while***, però s'utilitza ***for*** perquè és més abreviada.

   - Operacions sobre els elements

     El tractament de cada element consisteix a escriure'l pel canal de sortida i no es requereix cap altre tractament al final.

El programa resultant és el següent:

```
// Programa que escriu els n primers elements de la sèrie de Fibonacci

#include <iostream>
using namespace std;
int main(void){
int n;                                  // Nombre d'elements a tractar
int f,fanterior, fanterior2,i;          // Representació de la finestra: l'element i els 2 anteriors
    fanterior= 1; f = 1;
   cout<< "Introduiu el nombre d'elements de la serie de Fibonacci que voleu"
<<endl;
   cin>>n;
   if (n>1){
      cout<< "Els "<< n << "numeros de la serie de Fibonacci son:"<<endl;
      cout<< "1  1  ";    // S'escriuen els dos primers elements
      for (i=2;i< n;i++){
         fanterior2 = fanterior;          // fanterior passa a ser fanterior2
         fanterior = f;                   // f passa a ser fanterior
         f = fanterior + fanterior2;      // s'obté l'element següent sumant els 2 anteriors
         cout<<f<< endl;                  // S'escriu l'element generat
      }
    }                                     // fi de if (n>1)
  system("pause");
}
```

Programa 41

Programa que escriu el primer màxim local en una seqüència donada de nombres reals acabada en $0{,}0$.

La resolució d'aquest problema ha d'utilitzar l'estratègia de cerca per buscar el primer element de la seqüència que compleixi la propietat que defineix un màxim local, ser més gran que el predecessor i el successor. Es tracta d'un exemple de problema de finestra en el qual, a més de l'element de la seqüència que es vol tractar, s'han de considerar l'immediatament anterior i posterior.

1. **Definició i inicialització de variables**

   El tractament de cada element de la seqüència consisteix a comparar-lo amb l'anterior i el posterior; per tant, s'han de definir tres variables reals que representin els tres elements consecutius a la seqüència. A més, com que es tracta d'un problema de cerca, s'utilitza també una variable booleana, que representa en tot moment de l'execució si s'ha trobat l'element cercat. Les variables utilitzades són:

   - n, per guardar l'element que s'està tractant,
   - *nanterior*, per guardar l'element anterior a n,
   - *nposterior*, per guardar l'element posterior a n,
   - *trobat*, booleana que s'inicialitza a *false*.

   Fixeu-vos que en aquest exemple s'ha definit la constant FI, que representa la fi de la seqüència, el real $0{,}0$. Tot i que en programació l'ús de constants no és mai obligatori, és convenient quan un valor té un significat especial i es repeteix diverses vegades en el programa, com en aquest cas.

2. **Cerca d'un element a la seqüència**

   - Característiques de la seqüència

     - Primer element. Per tractar el primer element, s'han de llegir els tres primers reals que arriben pel teclat. S'ha de considerar el cas que la seqüència d'entrada tingui menys de tres elements; per això, només en el cas que el primer no sigui l'últim (el $0{,}0$) es llegirà un segon element, i només si aquest tampoc no ho és es llegirà un tercer element.

Si un programa té una instrucció de llegir i no s'hi introdueix cap valor pel canal d'entrada, es queda aturat.

     - Element següent. S'han d'actualitzar correctament les variables que representen els tres reals a considerar per tal de detectar-hi un màxim local. L'element ja tractat passa a ser l'element predecessor, l'element successor passa a ser l'element a tractar i l'element successor serà el següent que es llegirà pel teclat.

```
nanterior = n ;
n = nposterior;
cout << "Introduiu un real: ";
cin >> nposterior;
```

     - Darrer element. El que té el valor de la constant FI, és a dir, $0{,}0$.

   - Cerca

     Com que és un problema de cerca, mentre no es trobi un màxim local ni s'arribi al final de la seqüència s'ha de repetir el procés d'obtenir l'element següent i comprovar si és un màxim.

3. **Tractament final**

   Seguint l'estratègia de cerca, una vegada s'ha finalitzat el segon pas cal determinar si s'ha trobat l'element cercat (en aquest exemple, un màxim local) o no. A aquest efecte, s'utilitza l'estructura condicional *if*.

La resolució completa del programa es detalla a continuació.

//Programa que escriu el primer màxim local en una seqüència de nombres reals acabada en 0.

```
#include <iostream>
using namespace std;
const double FI=0.0;
int main(void) {
  double n, nanterior, nposterior;
  bool trobat=false;
  cout << endl << "Introduiu un nombre real: "; cin >> nanterior;
  if (nanterior != FI) {                              // si seqüència no buida
    cout << endl << "Introduiu un nombre real: "; cin >> n;
    if (n != FI) {
      cout << endl << "Introduiu un nombre real: "; cin >> nposterior;
      while (nposterior != FI && !trobat) {
        if (n> nanterior && n > nposterior) {
          trobat=true;
        } else {
```

```
            nanterior = n ;
            n = nposterior;
            cout << endl << "Introduiu un nombre real: "; cin >> nposterior;
          }
        }                                                    // fi del while
      }
      if (trobat) {
        cout << "El primer maxim local es "<< n << endl;
      } else {
       cout << "No s'ha trobat cap maxim local "<< endl;
      }
    } else {
      cout << "Sequencia buida"<< endl;
    }
     system("pause");
}
```

3.6.2 Problemes de seqüències de seqüències

Hi ha molts problemes que es poden veure com una seqüència de seqüències, és a dir, una seqüència d'elements per tractar en què el tractament de cada element requereix de l'aplicació de l'esquema de recorregut o cerca. Això implica que s'ha d'utilitzar l'esquema de recorregut o cerca per anar accedint als elements de la seqüència i l'esquema de cerca o recorregut per tractar cada un dels elements. A continuació, es presenta un exemple d'aquest tipus.

Programa 42

Programa que escriu els nombres primers d'una seqüència d'enters positius donada acabada en -1.

Com ja s'ha vist en altres exemples, el tractament de tots els elements d'una seqüència donada (introduïda pel canal d'entrada) implica la utilització de l'estratègia de recorregut, en què els elements s'obtenen llegint-los pel teclat i el procés acaba en trobar-ne el sentinella (en aquest cas, el -1).

El tractament de cada element consisteix a comprovar si és primer, i per a això s'utilitza la solució ja explicada per determinar si un nombre és primer. Aquesta solució consisteix a utilitzar l'estratègia de cerca per buscar si el nombre té algun divisor.

El programa resultant és el següent:

```
// Programa que escriu els enters que són primers d'una llista acabada en -1.

#include <iostream>
using namespace std;
int main(void) {
   bool trobat = false;
   int divisor,n;
   cout<< "Introduiu un enter positiu; per acabar -1: ";
   cin >> n;
   while (n!= -1){
      if (n == 1 ||n%2 == 0) {          // El número 1 i els parells no són primers
        if (n != 2)                     // El número 2 per definició és primer
           {trobat = true;}
```

```
        else { cout << "El numero 2 es primer "<<endl; }
      } else {
       trobat = false;
       divisor=3;
       while (!trobat && divisor*divisor<= n) {
          if (n % divisor == 0) {
             trobat = true;
           } else{
            divisor = divisor +2;
            }
        }                                        // fi del while intern
     }
  if (!trobat) cout <<"El numero "<< n <<"es primer"<<endl;
   cout<< "Introduiu un enter positiu; per acabar, -1: ";
   cin >> n;
}                                              // fi del while principal
system("pause");
}
```

3.7 Exercicis proposats

3.7.1 Exercicis bàsics

1

Programa que, donada una seqüència d'enters acabada en 0, en què cada enter representa una quantitat en pessetes, escrigui per pantalla la seqüència de les quantitats entrades en euros.

2

Programa que calculi la mitjana d'una seqüència de nombres enters acabada en zero, introduïda pel teclat.

3

Programa que, donada una seqüència d'enters acabada en 0 introduïda pel teclat, determini si hi ha més enters parells que imparells.

4

Programa que escrigui el nombre de repeticions del primer real d'una seqüència de nombres reals acabada en 0,0, introduïda pel teclat.

5

Programa que obté el màxim enter positiu en una seqüència d'enters acabada en 0, introduïda pel teclat. El programa ha de comprovar que cada element de la seqüència sigui un enter (no un real) i positiu.

6

Programa que compti el nombre de vocals i el nombre de consonants d'una frase acabada en punt, introduïda pel teclat.

7

Programa que, donada una x introduïda pel teclat, calculi e^x a partir del desenvolupament de la sèrie de Taylor, $e^x = 1 + x + x^2/2 + \cdots + x^k/k!$, on la condició d'acabament és que l'últim terme de la sèrie calculat sigui menor a $0{,}03$.

8

Programa que calculi el mínim valor absolut de la funció $f(x) = 3x - 4$ i la seva abscissa, donats un interval i una precisió èpsilon, introduïts pel teclat.

3.7.2 Exercicis avançats

9

Programa que, donada una seqüència d'enters acabada en 0 i introduïda pel teclat, determini si correspon a una seqüència decreixent.

10

Programa que determini si hi ha un número igual a la suma de l'anterior i el successor en una seqüència d'enters acabada en 0, introduïda pel teclat.

11

Programa que calculi el perímetre d'un polígon representat per una seqüència dels seus vèrtexs, en què cada vèrtex es representa com dos reals (x, y) i la seqüència finalitza amb la segona aparició del primer vèrtex.

12

Programa que determini si tots els vèrtexs d'un polígon estan al mateix quadrant. El polígon es representa com una seqüència de vèrtexs en què cada un es representa com dos reals (x, y) i finalitza amb la segona aparició del primer vèrtex.

13

Programa que determini la posició de la primera paraula amb igual nombre de vocals que de consonants.

14

Programa que compti quants caràcters hi ha abans de la primera aparició de l'article "la" en una frase acabada en punt, introduïda pel teclat.

15

Programa que compti els trams creixents en una seqüència d'enters acabada en 0, introduïda pel teclat.

4 Funcions

4.1 Introducció

Una funció és una successió d'instruccions que realitzen una tasca determinada, a la qual s'assigna un nom. Aquesta seqüència d'instruccions s'executa cada vegada que el nom de la funció s'invoca.

Les operacions que realitza una funció són sempre les mateixes, però les dades que usa canvien cada vegada que es crida la funció. L'ús de funcions ens permet estructurar els programes C++ en forma de mòduls. Les funcions d'ús més freqüent acostumen a definir-se en biblioteques.

Qualsevol programa escrit en C++ conté una o més funcions. Una d'aquestes funcions ha de dir-se ***main***, i opcionalment pot tenir paràmetres. Un programa escrit en C++ es comença a executar a partir de la funció ***main***. Aquesta funció pot invocar altres funcions, tant les definides pel programador com les que es troben a les biblioteques estàndard del compilador de C++. Les sentències descrites dins d'una funció comencen a executar-se quan el nom de la funció s'invoca des del programa principal (***main()***) o des d'una altra funció. Normalment, quan es crida una funció també se li proporciona un conjunt de dades (arguments), que s'usen quan s'executen les instruccions de la funció.

4.2 Definició i crida

La definició d'una funció consta de dues parts:

- Una línia anomenada *capçalera*, on s'indiquen el nom de la funció, el tipus del resultat que retorna i, opcionalment, els paràmetres que utilitza.
- Una successió d'instruccions tancada entre claus que donen cos a la funció.

```
tipus nomfuncio ([paràmetre 1], ... ,[paràmetre N])
{instruccions}
```

on:

- El tipus especifica el tipus de dades del valor que retornarà la funció mitjançant la instrucció ***return***.
- Cada paràmetre consta d'un tipus, seguit d'un identificador. Els paràmetres s'han de separar amb comes i tancar-se entre parèntesis. Els paràmetres són opcionals.
- Les sentències formen el cos de la funció i es delimiten amb claus.

Quan una funció es defineix després del punt en què es realitza la crida, es necessita un prototip de la funció que es vol invocar. El prototip consta únicament de la capçalera de la funció. Un prototip és una declaració de la funció que informa el compilador sobre el tipus del resultat que retorna la funció i els seus paràmetres. El prototip és també la interfície que la funció presenta a la resta de les funcions. Així, si altres funcions volen utilitzar-la, només han de conèixer el seu prototip i adaptar-hi les crides. És a dir, han de passar el nombre d'arguments especificat al prototip amb el tipus de dades adequat.

Les variables que s'usen en la declaració o en la definició d'una funció es diuen *paràmetres*.

El terme *arguments* es reserva per als valors concrets que es donen als paràmetres en les crides a una funció. La crida d'una funció es realitza escrivint el seu nom i una llista d'arguments (en cas que siguin necessaris), delimitats per parèntesis. La quantitat d'arguments i el tipus de cadascun han de coincidir amb la definició de la funció o del prototip. Els noms dels arguments no són necessàriament els mateixos que els noms de les variables usades per definir els paràmetres de la funció en la seva definició.

Programa 43

Funció que calcula la distància entre dos punts de la recta real.

```
/* Zona d'inclusions */
#include <iostream>
 using namespace std;
/* Zona de prototips */
double distancia(double x, double y);
/* Zona de definició de la funció principal main() */
int main() {
   double a=2.3, b=-33.3;
   double d=distancia(a,b);
   cout << "La distancia entre " << a << " i " <<
    b << " es " << d << endl;
   system("pause");
}
/* Zona de definició de funcions auxiliars */
/* Funció: distància
   Ús:   f = distància(x,y);
   ----------------------------------------------------
   Aquesta funció retorna la distància entre x i y.   */
double distancia(double x, double y) {
  double d=x-y;
  if (d<0) d=-d;
  return d;
}
```

Com qualsevol altre programa escrit en C++, el programa comença amb la funció ***main()***. La funció ***main*** comença amb la declaració següent de les variables locals.

```
double a=2.3, b=-33.3;
```

Els valors que apareixen en la crida d'una funció s'anomenen *arguments*. Els paràmetres són els identificadors utilitzats en la definició de la funció. Aquests identificadors només estan disponibles durant l'execució de la funció. En aquest exemple, ***main*** crida la funció distància amb la llista següent d'arguments `(a,b)`. El paràmetre ***x*** rep l'argument ***a*** i el paràmetre ***y*** l'argument ***b***. La funció ***main*** passa el control a la funció que crida. La instrucció següent que s'executa és la primera instrucció de la funció. La funció retorna

el control a *main* mitjançant la instrucció *return*. La instrucció *return* retorna el control del programa i el valor de la distància entre *x* i *y* a la funció que va invocar a *"distancia"*. La funció que fa la crida pot ignorar el valor retornat per la funció invocada. En aquest exemple, *main* guarda aquest valor en la variable *d* i l'escriu per pantalla.

4.3 Pas de paràmetres

Els arguments d'una funció es poden passar *per valor* o *per referència*. Les instruccions de la funció no operen amb l'argument que s'ha passat per *valor*, sinó amb una còpia local d'aquest que s'elimina un cop finalitzada l'execució de la funció. És a dir, l'argument no queda alterat com a resultat de l'execució de la funció. Quan es passen els arguments per *referència*, les instruccions de la funció operen directament amb l'argument que es passa.

Habitualment, les funcions fan còpies locals dels seus arguments, és a dir, el pas d'arguments és per valor. Per aconseguir que una funció operi sobre una variable en lloc d'operar sobre la seva còpia, s'ha de passar la seva d'adreça. Els canvis que es facin sobre les dades usant la seva adreça persistiran després de l'execució de la funció. Els paràmetres que es passen per referència es marquen amb el símbol & just després del tipus, en la primera línia de la definició de la funció i del seu prototip, però no en el cos ni en les crides que es facin dins de la mateixa funció.

4.4 Àmbit d'identificadors

Definim l'*àmbit* d'un identificador (variable, funció ...) com el conjunt de sentències en què podem utilitzar-lo. Cada identificador es defineix dins el codi mitjançant una declaració i dins d'aquest àmbit no hi pot haver una altra declaració amb el mateix identificador. L'àmbit es correspon amb una zona del codi font englobada entre claus { } (bloc), amb una llista de paràmetres dins una funció o amb l'espai d'una unitat de compilació no inclosa en cap altre àmbit.

Les regles per al càlcul de l'àmbit d'un identificador són les següents:

1. Un identificador declarat dins un bloc és accessible únicament des d'aquest bloc i tots els blocs inclosos dins ell (es considera local al bloc). Un paràmetre formal es considera també una declaració *local* al bloc de la funció.
2. Els identificadors declarats fora de qualsevol bloc es consideren *globals* i es poden utilitzar des de qualsevol bloc de la unitat de compilació.
3. Quan tenim un bloc dins un altre bloc i en tots dos declarem identificadors amb el mateix nom, l'identificador del bloc intern *oculta* el del bloc extern.

Programa 44

```
/*****************************************************/
/* Programa exemple d'àmbit utilitzat correctament   */
/*****************************************************/
#include <iostream>
using namespace std;
int g; //variable global
int sumar(int x, int y);
int main()
{
int suma; // variable local dins main
```

```
g = 12;
suma = sumar(2,3);
cout << "suma=" << suma << " g=" << g << endl;
// presenta els valors suma=5 g=12
system("pause");
}
int sumar(int x, int y)
{
int g;  // variable local dins de sumar
// "oculta" la variable g global
g = x + y;
return g;
}
```

En una funció, només s'han d'utilitzar variables locals a ella. Les variables globals no s'han d'utilitzar mai. D'aquesta manera, s'independitza la funció del programa o bloc des del qual fem la crida. La independència permet que sigui més fàcil fer modificacions en el programa, que la funció es pugui reutilitzar en altres programes i també que sigui més fàcil treballar en equip. Si no se segueixen aquestes regles, es poden produir efectes no volguts. Vegem-ne uns exemples:

```
/*****************************************************/
/* Programa exemple d'àmbit utilitzat incorrectament    */
/*****************************************************/
#include <iostream>
using namespace std;
int g;  // Variable global
int sumar(int x, int y);
int main()
{
int suma;  // variable local dins del main
g = 12;
suma = sumar(2, 3);
cout << "suma=" << suma << " g=" << g << endl;
// presenta els valors suma=5 g=5
// S'ha modificat 'misteriosament' el valor de g
system("pause");
}
int sumar(int x, int y)
{
g = x + y;
return g;
}
```

4.5 Funcions i accions

Formalment, *les funcions* es caracteritzen perquè sempre retornen un valor. Tota definició d'una funció conté almenys una instrucció del tipus *return expressió*. Les funcions generen expressions, és a dir, dades. Malgrat que el compilador ho permeti, no es recomana que els arguments d'una funció es passin per referència. Les *accions* no retornen cap valor i accepten arguments que es poden passar per valor o per referència. Les accions són, per tant, instruccions complexes que produeixen efectes (per exemple, mostrar una dada o un dibuix a la pantalla) i/o modifiquen l'estat del programa.

4.5.1 Accions

La capçalera d'una acció comença amb la paraula *void*, que indica que no retorna cap valor. A continuació, hi ha un exemple en el qual es defineix i es realitza una crida a una acció.

Programa 45 *Intercanvi dels valors de dues variables. L'acció* **"intercanvia"** *utilitza dos paràmetres d'entrada/sortida. El pas de paràmetres en aquesta acció es realitza per referència. Això s'indica utilitzant* & *en la declaració dels paràmetres de l'acció.*

```
#include <iostream>
using namespace std;
void intercanvia (double& a, double& b);
int main () {
   double x=2.3, y=-33.3;
   cout << "x = " << x << "   i = " << y << endl;
   intercanvia(x, y);
   cout << "x = " << x << "   i = " << y << endl;
   system("pause");
}
void intercanvia (double& a,  double& b) {
    double temp;
    temp = a;   a = b;   b = temp;
}
```

Els paràmetres de l'acció ***intercanvia*** són referències a variables de tipus *double*. Quan es realitza el pas d'arguments per referència, es còpia l'adreça dels arguments als paràmetres. Així, les variables x i a indiquen la mateixa adreça de memòria, i les variables b i y indiquen la mateixa adreça de memòria. És a dir, els paràmetres de l'acció (les variables a i b) són referències a les variables utilitzades com a arguments de la crida (les variables x i y).

4.5.2 Funcions

Les funcions sempre retornen un valor del tipus especificat a la capçalera. Es recomana passar sempre els paràmetres per valor, no per referència. Un cas particular de funcions són aquelles que retornen valors del tipus booleà (veritat/fals). Aquestes funcions es solen anomenar *predicats*.

Programa 46 *Programa que rep un nombre enter i diu si és un nombre primer.*

```
#include <iostream>
using namespace std;
bool es_primer(int n);
int main() {
   int n;
   cout << "Introduiu un nombre enter positiu: " << endl;
   cin >> n;
   if (es_primer(n))
        cout << n << " es primer" << endl;
   else
        cout << n << " no es primer" << endl;
   system("pause");
}
/* Zona de definició de funcions auxiliars */
/* Funció: es_primer
```

```
   Ús:    b = és_primer(n);
   -------------------------------------------------------
   Aquesta funció estableix si el nombre n és primer.
   Utilitza el fet que, si n té un divisor diferent de 1 i n,
   llavors n ha de tenir un divisor menor o igual que
   l'arrel quadrada de n. */
bool es_primer(int n) {
    bool primer;
    int d;
    if (n==1) {
        primer=false;
    } else {
        primer=true;
        d=2;
        while (primer && d*d<=n) {
            if (n%d==0)
              primer=false;
            else
              d++;
        }
    }
    return primer;
}
```

En la definició de la funció, s'especifica el tipus de valor de retorn *bool*, que indica que el valor retornat de la instrucció *return* serà fals o veritable.

```
bool es_primer(int n)
```

El tipus de la variable local de la funció *es_primer*, que utilitzem per emmagatzemar el resultat d'establir si el nombre *n* és primer o no, ha de ser de tipus *bool* per coincidir amb la definició.

```
bool primer;
```

4.6 Exemples

Programa 47 *Definició i utilització de dues funcions que calculen l'àrea d'un triangle.*

- *Si coneixem dos dels seus costats i els angles compresos entre ells.*
- *Si coneixem dos dels angles i el costat que els uneix.*

S'ha d'usar una acció que permeti escollir els valors a partir dels quals es vol calcular l'àrea.

```
#include <iostream>
#include <cmath>
using namespace std;
const double PI=3.14159265;
double radian(double ang);
double area1(double b, double c, double a);
double area2(double b, double c, double a);
double area3(double b, double c, double a);
int EscullMode(int Mode);
```

```
int main() {
  int Mode=9;
  do {
     system("cls");
     cout << "Mode a utilitzar per calcular l'area: "<<endl;
     cout << "1 - Es coneixen dos costats i l'angle entre ells "<<endl;
     cout << "2 - Es coneix un costat i els dos angles adjacents "<<endl;
     cout << "3 - Es coneixen els tres costats "<<endl;
     cout << "9 - Sortir "<<endl<<endl;
     cout << "Opcio: ";
     cin >> Mode;
     Mode = EscullMode(Mode);
     } while (Mode!=9);
     system("pause");
}
/* es passen de graus a radians */
double radian(double ang){
       return (ang * PI/180);
}
/* Càlcul de l'àrea segons mòdul 1 */
double area1(double b, double c, double A){
return 0.5*(b*c)*sin(radian(A));
}
/* Càlcul de l'àrea segons mòdul 2 */
double area2(double B, double c, double A){
double C=180-(A+B);
double num=pow(c,2)*sin(radian(A))*sin(radian(B));
double den=sin(radian(C));
return 0.5*(num/den);
}
/* Càlcul de l'àrea segons mòdul 3 (Fórmula d'Heron) */
double area3(double b, double c, double a){
return sqrt((a+b+c)*(a+b-c)*(b+c-a)*(c+a-b))/4;
}
/* Càlcul de l'àrea en funció del mòdul */
int EscullMode(int Mode) {
double l1,l2,l3,a1,a2;
char dummy;
  if (Mode==9) return 9;
  if (Mode==1)   {
    cout << "Costat 1: ";
    cin >> l1;
    cout << "Costat 2: ";
    cin >> l2;
    do
    {
      cout << "Angle (mes gran que 0 i menor que 90): ";
      cin >> a1;
    } while(a1<=0 || a1>90);
   cout << endl <<"Area: "<<area1(l1,l2,a1)<<endl;
  }
  if (Mode==2)   {
    cout << "Introduiu el valor d'un dels costats: ";
```

```
    cin >> l1;
    l1 = abs(l1);
    do
    {
      cout<<"Introduiu els valors dels angles: A i B, A + B < 180 "<<endl;
      cout << " Angle A ";cin >> a1;
      cout << " Angle B ";cin >> a2;
    } while(a1+a2>=180);
    cout << endl << "Area: " << area2(a2,l1,a1) <<endl;
  }
  if (Mode==3)   {
    cout << "Costat 1: ";
    cin >> l1;
    cout << "Costat 2: ";
    cin >> l2;
    cout << "Costat 3: ";
    cin >> l3;
    cout << endl <<"Area: "<<area3(l1,l2,l3)<<endl;
  }
   cout << "Per continuar, premi s: " ;
   cin >> dummy;
   if (dummy == 's' || dummy == 'S') return 0;
   return 9;
}
```

S'usen les funcions següents de la biblioteca *cmath:* $\sin(\gamma)$ i $\cos(\gamma)$, on γ està expressada en radians.

Programa 48 *Definició i utilització d'un predicat (funció que retorna un valor booleà) que comprovi si una data descrita mitjançant tres enters (dia, mes i any) és correcta.*

```
#include <iostream>
using namespace std;
bool testdia(int d, int m, int a);
bool traspas(int a);
int main() {
  int d,m,a;
  do  {
  cout << "Introduiu l'any (entre 1 i 9999): ";
  cin >> a;
  }
  while(a<=0 ||a >9999);
  do  {
  cout << "Introduiu el mes (entre 1 i 12): ";
  cin >> m;
  }
  while (m<1 || m>12);
  do  {
  cout<<"Introduiu el dia (entre 1 i 31, segons el mes): ";
  cin>>d;
  }
  while (testdia(d,m,a)==false);
  cout<<endl;
  cout<<"la data es correcta "<<d<<" "<<m<<" "<<a<<endl;
```

```
  system("pause");
}

bool testdia(int d,int m,int a)
{
   if  ((m==4 || m==6 || m== 9 || m==11) && (d>30)) return false;
   else
      if (m==2)  {
         if (traspas(a) && d>29) return false;
         if (!traspas(a) && d>28) return false;
      }
      else    {
         if(m==1||m==3||m|5 ||m==7 ||m==8||m==10 ||m==12) {
            if(d>31) return false;
         }
      }
   return true;
}
bool traspas(int a)
{
   if ((a % 4) == 0)   {
      if ((a % 100) == 0)  {
         if ((a % 400) == 0) return 1;
         else return 0;
      }
      else return 1;
   }
   else return 0;
}
```

Programa 49 *Definició i utilització d'una acció que intercanvia, en cas necessari, els valors de les variables que es passen com a arguments, de forma que els valors d'aquestes variables quedin en ordre creixent.*

```
#include <iostream>
using namespace std;
void intercanvia(double& x, double& y);
void ordena_creix(double& p, double& s, double& t);
int main() {
   double primera, segona, tercera;
   cout << "Introduiu tres nombres: ";
   cin >> primera >> segona >> tercera;
   ordena_creix(primera,segona,tercera);
   cout << "Els nombres introduits "<<
   "ordenats creixentment son " <<
   primera << ", " << segona << ", " << tercera << endl;
   system("pause");
}
void intercanvia(double& x, double& y) {
  double aux;
  aux=x; x=y; y=aux;
}
void ordena_creix(double& p, double& s, double& t) {
```

```
  if (p>s)
    intercanvia(p,s);
  if (s>t) {
    intercanvia(s,t);
    if (p>s) {
       intercanvia(p,s);
    }
  }
}
```

Programa 50 *Aquest programa calcula la suma dels dígits d'un nombre enter positiu introduït per pantalla.*

```
#include <iostream>
using namespace std;
int suma_dig(int n);
int main() {
   int nombre;
   cout << "Aquest programa llegeix un nombre "<<
   "i calcula la suma dels seus digits." <<endl << endl;
   cout << "Introduiu un nombre enter positiu: ";
   cin >> nombre;
   cout << endl;
   cout << "La suma dels digits del nombre " <<
   nombre << " es " <<suma_dig(nombre) <<
    "." << endl;
   system("pause");
}
int suma_dig(int n) {
   int suma=0;
   if (n < 0) n=-n;
   while (n!=0) {
         //Sumem l'última xifra, que s'obté
         //calculant el mòdul 10 del nombre.
         //Per exemple, l'última xifra de 12345 és
         //12345 % 10 = 5
   suma+=n%10;
         //Repetim el procés per al nombre sense
         //l'última xifra, que s'obté calculant
         //el quocient de dividir el nombre original
         //per 10. Per exemple, 12345/10 =1234.
    n=n/10;
   }
   return suma;
}
```

Programa 51 *Fer un programa que utilitzi una funció per calcular el factorial d'un nombre. El nombre l'ha d'introduir l'usuari per pantalla i ha de ser un enter positiu. El factorial de* n *(i.e.,* n!*) està definit per:* `n! = n · (n-1) ··· 1` *on* `0! = 1` *i* `1! = 1`.

```
#include <iostream>
using namespace std;
```

```
int factorial(int n);
int main() {
   int nombre;
   cout << "Introduiu un nombre: " << endl;
   cin >> nombre;
   cout << endl << "El factorial de " << nombre << " es " <<
           factorial(nombre) << endl;
   system("pause");
}
/* Zona de definició de funcions auxiliars */
/*
* Funció: factorial
* Ús:    f = factorial(n);
* ------------------------------------------------
* Aquesta funció retorna el factorial f del nombre n.
*/
int factorial(int n) {
   int resultat=1;
   if (n>1) {
    for (int i=2; i<=n; i++)
          resultat*=i;   //abreviatura de resultat=resultat*i;

   }
   return resultat;
}
```

Programa 52 *Fer una funció que converteixi un nombre en base* 10 *a base* 2.

```
#include <iostream>
using namespace std;
int dec2bin(int n);
int main() {
   int num;
   cout << "Introduiu un nombre enter: " << endl;
   cin >> num;
   cout << endl;
   cout << "El nombre " << num <<
   " en base 10 equival al nombre " <<
   dec2bin(num) << " en base 2." << endl;
   system("pause");
}
int dec2bin(int n) {
   int n_binari=0, coef=1;
   while (n!=0) {
      n_binari+=coef*(n%2);
      coef*=10;
      n/=2;
   }
   return n_binari;
}
```

4.7 Exercicis proposats

4.7.1 Exercicis bàsics

1

Definiu una funció que determini si un nombre de 5 xifres és capicua. És a dir, si la seva primera i última xifra són iguals, i si la seva segona i quarta xifra són iguals. Utilitzeu les operacions de divisió entera per 10 i la resta mòdul 10 per extreure les diferents xifres del nombre. Utilitzeu correctament aquesta funció cridant-la des de la funció *main* d'un programa. **[r]**

2

Definiu una funció que calculi el nombre resultant d'escriure a l'inrevés les xifres d'un nombre enter. Per exemple, `inreves(321) = 123`. Utilitzeu aquesta funció correctament en un algoritme. **[r]**

3

Definiu un predicat que estableixi si un nombre és capicua. Utilitzeu la definició de la funció "inrevés" de l'apartat anterior. Utilitzeu aquesta funció correctament en un algoritme que interaccioni amb l'usuari. **[r]**

4

Definiu una funció que determini si un nombre enter positiu és primer utilitzant les idees següents: *a)* Si un nombre té un divisor parell, llavors ha de ser divisible per 2; *b)* si un nombre té un divisor imparell, aquest s'ha de trobar entre 3 i $n^{1/2}$. El vostre algoritme ha de preveure que l'únic nombre primer parell és el 2, i que el nombre 1 no és primer. Utilitzeu aquesta funció correctament en un algoritme. **[r]**

5

Definiu una funció que calculi la representació en base n (on n és un nombre de 2 a 9) d'un nombre a partir de la seva representació en base 10. Per exemple, `dec2base(10,3) = 101`. Utilitzeu aquesta funció correctament en un algoritme que interaccioni amb l'usuari. **[r]**

6

Definiu una funció que calculi la representació en base 10 d'un nombre a partir de la seva representació en base n. Per exemple, `bin2dec(1010) = 10`. Utilitzeu aquesta funció correctament en un algoritme que interaccioni amb l'usuari. **[r]**

7

Construïu un programa que calculi el mínim i el màxim d'una llista de N nombres enters introduïts per l'usuari utilitzant les funcions de mínim i màxim. **[r]**

4.7.2 Exercicis avançats

1. Enunciat 1: Cercle i quadrat

   Programa que calcula el costat L del quadrat de superfície aproximadament igual al d'un cercle de radi R. Es demanen com a dades d'entrada el valor de R i la diferència màxima permesa (D) entre les dues superfícies. El valor de D pot ser tan petit com es vulgui.

   Utilitzarem les funcions següents:

- double `calcula_area_cercle` (double R): calcula l'àrea del cercle de radi R
- double `calcula_area_quadrat` (double L): calcula l'àrea del quadrat de costat L
- double `diferencia_area` (double R, double L): obté la diferència entre l'àrea del cercle de radi R i el quadrat de costat L.

Dins la funció *main*, aplicarem l'algorisme següent:

Partint d'un quadrat de costat $L = R$ i del valor inicial de la variable `inc=R/2`, incrementem o disminuïm, respectivament, L en una quantitat `inc`, depenent del signe (positiu o negatiu) del valor de retorn de la funció `diferencia_area`. Cada cop que detectem un canvi de signe entre el valor actual i el valor anterior de la diferència, dividim la variable `inc` entre 2. El programa finalitza quan el valor absolut de la diferència entre les àrees és menor o igual a D.

2. Enunciat 2: Moviment parabòlic

Suposem que tenim 2 llançadors de boles separats una distància de 100 metres. De forma simultània, disparem una bola de cada llançador amb angles θ_1 $(0 < \theta_1 < \pi/2)$ i θ_2 $(\pi/2 < \theta_2 < \pi)$, i amb velocitats V_1 i V_2 respectivament (figura 4.1). Es tracta de dissenyar un programa que calculi la distància entre els dos punts de cada trajectòria en el moment en què s'assoleix l'alçada màxima. Suposarem que no hi ha fregament i considerem les boles com a punts sense dimensió. En aquestes condicions, podem aplicar les equacions del moviment parabòlic:

$$X = V_x \cdot t$$
$$Y = V_y \cdot t - (g \cdot t^2)/2$$

Essent: $V_x = V \cdot \cos(\theta)$, on θ està en radians
$V_y = V \cdot \text{sen}(\theta)$
$g = 9{,}81$

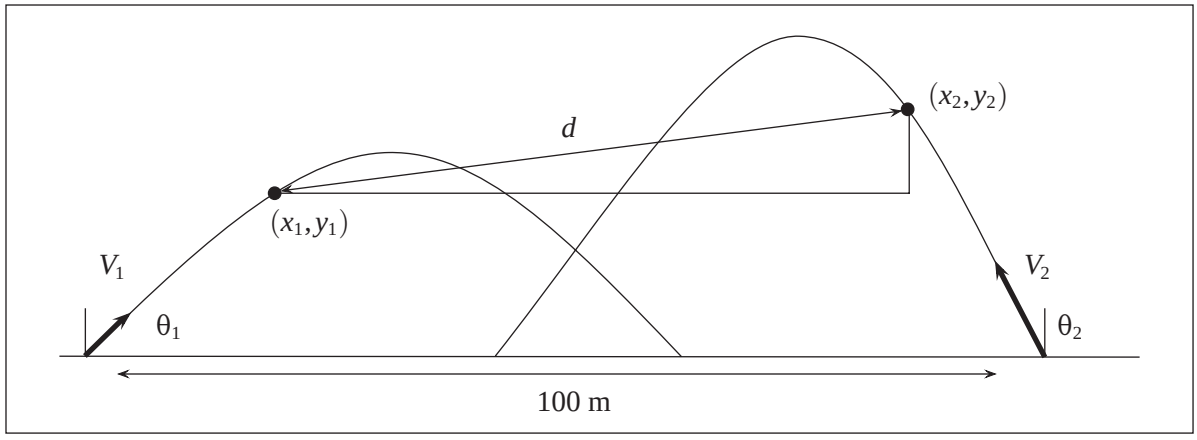


Fig. 4.1 Moviment parabòlic

La distància entre les dues boles a cada posició de la trajectòria és $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Per trobar el punt corresponent a l'alçada màxima, hem de calcular el màxim local de cada trajectòria, i per això hem de considerar una *finestra* de 3 punts: el punt (x,y), l'anterior (x_a,y_a) i el posterior (x_p,y_p). Tindrem un màxim local a (x,y) si $y_a < y$ i $y > y_p$. Hem de tenir en compte que el valor de 'y' no pot ser negatiu (el moviment d'una bola finalitza quan la coordenada 'y' d'aquesta agafa el valor 0 després de l'instant inicial). Considerem intervals de temps de 0,001 segons per a calcular cada punt de la trajectòria.

4.7.1.1 Tasques per realitzar

- Definiu com a constants les expressions següents:

  Intervals de temps: `interval = 0,001`.
  El valor de π `PI = 3,14159265`.
  La força de gravetat: `g = 9,81`.

- Demaneu a l'usuari la introducció dels valors següents:

  Per a la bola 1: la velocitat inicial V_1 major que zero i l'angle en graus $(0 < \text{angle} < 90)$.
  Per a la bola 2: la velocitat inicial V_2 major que zero i l'angle en graus $(90 < \text{angle} < 180)$.

  Definiu les funcions i accions següents amb els seus prototipus respectius:

  - Funció que converteix un angle de graus a radians double radian (double ang)

  - Funció que calcula les components V_x, V_y de cada bola (V_{1x}, V_{1y}, V_{2x}, V_{2y}) utilitzant les fórmules següents (els angles s'expressen en radians):

$$V_x = V \cdot \cos(\theta)$$
$$V_y = V \cdot \text{sen}(\theta)$$

  - Acció que calcula la trajectòria (posicions XY) en funció del temps. Suposem que l'origen de coordenades es troba en el llançador 1. En el moment en què una bola torna a *'tocar terra'* $(Y = 0)$, mantenim la seva posició invariable.

```
void trajectoria(double t, double Vx, double Vy, double& X, double& X)
```

  - Funció que retorna la distància entre les dues boles.

```
double distancia(double X1, doble Y1, double X2, double Y2);
```

  - Definiu un programa *main* que:

    - Demani l'entrada de valors a l'usuari.
    - Calculi la posició de cada bola en el primer interval de temps.
    - Mentre alguna de les ordenades Y de les boles sigui 0, calculi la trajectòria. S'han de guardar els valors de les coordenades de cada bola en què s'hagi detectat un màxim local.
    - Calculi la distància corresponent als màxims locals de cada bola i en presenti el resultat.

4.8 Recursivitat

En general, una definició és recursiva si el concepte definit forma part de la pròpia definició. Aplicant aquesta definició a les funcions, diem que una funció és recursiva si la pròpia funció conté una o més crides a si mateixa.

Una funció pot ser recursiva de foma directa (si es crida a si mateixa) o indirecta (si crida una altra funció que posteriorment la torna a cridar). En aquest capítol, ens centrarem en les funcions recursives directes. Quan definim una funció recursiva, cal anar amb compte amb el problema de la recursió infinita (bucle infinit).

Aquest problema apareix quan una funció es crida a si mateixa un nombre no limitat de vegades i té com a consequència que el programa no finalitza. Per tal d'evitar aquest problema, cal tenir en compte que tota funció recursiva necessita una condició d'aturada, anomenada també *cas base* o *trivial*. Definim *cas base* com aquella combinació d'arguments amb els quals la crida a la funció retorna un resultat sense necessitat de tornar-se a cridar a si mateixa. Si la funció recursiva es crida en un cas més complex, les crides successives a si mateixa aniran descomponent aquest cas en casos més sencills fins a arribar al cas base i determinar el resultat final de la funció.

Hi ha molts algorismes que només es resolen utilitzant la recursivitat o, si més no, la solució més directa i elegant es basa en la utilització de funcions recursives. Per exemple, el recorregut de determinades estructures de tipus arbre s'acostumen a resoldre de manera recursiva per assegurar-nos de passar per totes les branques de l'arbre.

Així doncs, en definir una funció recursiva, cal tenir presents els punts següents:

1. *Cas base:* Determinar el cas o els casos en què retorna un valor sense necessitat de cridar de nou a la funció.
2. *Casos recursius:* Considerar les crides successives utilitzant la inducció.
3. *Finalització:* Raonar sobre l'aturada del procés recursiu, indicant els valors que agafen els arguments en cada nova crida.

4.9 Exemples de recursivitat

Programa 53 *Per entendre millor el concepte de funció recursiva n'exposem un cas sencill: càlcul de la potència sencera d'un nombre sencer* $b^e (e >= 0)$. *Aquesta funció ha de retornar com a resultat un nombre sencer i acceptar com a arguments dos nombres sencers.*

1. Cas base: $e = 0$. *En aquest cas, tenim que* $b^0 = 1$. *El resultat d'aquest càlcul és directe.*
2. Cas recursiu: *Utilitzem la igualtat* $b^e = b \cdot b^{e-1}$, *que descompon el càlcul de* b^e *pel càlcul de* b *per* b^{e-1}.
3. Finalització: *A cada crida recursiva, el valor de l'argument que representa l'exponent ha de decréixer en una unitat. En tractar-se d'un nombre natural, arribarà un moment en què el valor d'aquest argument será zero, amb la qual cosa ja no haurà de tornar a cridar a la funció.*

```
#include <iostream>
using namespace std;
int pot(int base, int exponent);
int main() {
int b=3, e=2;
cout << pot(b,e) << endl;
system("pause");
}
int pot (int base, int exponent) {
if (exponent == 0) //cas base
     return 1;
else // cas recursiu
     return base * pot(base, exponent-1);
// A cada crida, el segon argument (exponent) decreix.
// En algun moment arriba al valor 0 i finalitzen les crides recursives.
}
```

Analitzem ara la següent crida "*pot(3,2)*". La funció definida executa la instrucció

```
return 3 * pot(3,1);
```

El control no torna a *main* a causa de la nova crida a "*pot*" dins l'argument de la instrucció *return*. Aquesta segona crida executa la instrucció

```
return 3 * pot(3,1);
```

Es produeix una nova crida recursiva a la funció *pot*. Aquesta tercera crida executa la instrucció.

```
return 1;
```

Aquesta instrucció retorna el valor 1 i el control, però no a *main*, sino a la crida que es va fer a la funció *pot* amb els arguments $(3,1)$. Aquesta darrera funció executa la instrucció `return 3*1`, retornant el valor 3 i el control a la crida que es va fer a la funció *pot* amb els arguments $(3,2)$. Finalment, la funció retorna el valor 9 i el control a '*main*' amb la instrucció

```
return 3*3;
```

Programa 54 *Factorial d'un nombre natural* $(n!)$*. Considerem els 3 apartats de la definició d'una funció recursiva:*

1. Cas base: *Si* $n = 0$*, llavors* $n! = 1$.
2. Cas recursiu: *Si* $n > 0$*, llavors* $n! = n \cdot (n-1)!$.
3. Finalització: *A cada crida recursiva, l'argument de la funció decreix en una unitat. Això significa que en algun moment tindrà el valor* 0*, cosa que finalitzarà les crides recursives.*

```
#include <iostream>
using namespace std;
int factorial(int n);
int main() {
int nombre;
cout << "Introduiu un nombre: ";
cin >> nombre;
cout << nombre << " = " << factorial(nombre)<<endl;
system("pause");
}
int factorial(int n) {
if (n == 0) // cas base
      return 1;
else return n * factorial(n-1); // cas recursiu
}
```

Programa 55 *Càlcul del* n*-èsim element de la successió de Fibonacci, que definim de la manera següent:* $Fib(0) = 0$*,* $Fib(1) = 1$*,* $Fib(n) = Fib(n-1) + Fib(n-2)$*, on* $Fib(n)$ *significa el terme* n*-èsim de la successió.*

1. Casos base: $Fib(0) = 0$ *y* $Fib(1) = 1$.
2. Cas recursiu: *Si* $n > 1$*, llavors* $Fib(n) = Fib(n-1) + Fib(n-2)$.

3. Finalització: *A cada crida recursiva, l'argument de la funció decreix en una o dues unitats. Això significa que en algun moment tindrà el valor* 0 *o* 1*, cosa que finalitzarà les crides recursives.*

```
#include <iostream>
using namespace std;
int fib(int n);
int main() {
int nombre;
cout << "Introduiu un nombre natural: "<< endl;
cin>>nombre;
cout << "El " << nombre << "-esim nombre de Fibonacci es: " <<
fib(nombre) << endl;
system("pause");
}
int fib(int n) {
if (n==0) return 0; // cas base
else if (n==1 or n==2) return 1; // cas base
else return fib(n-1) + fib(n-2); // cas recursiu
}
```

Programa 56 *Càlcul de les combinacions de* n *elements, agafats de* m *en* m*. És a dir, el nombre de maneres diferents en què es poden escollir* m *elements entre un conjunt de* n *elements. Considerem els tres apartats de la definició d'una funció recursiva.*

1. Casos base: *Si* $n = m$ *o* $m = 0$*, llavors* $\binom{n}{m} = 1$.
2. Cas recursiu: *Si* $n > m$ *y* $m > 0$*, llavors* $\binom{n}{m} = \dfrac{n!}{m! \cdot (n-m)!}$*, que reescrivim de la manera següent:* $\binom{n}{m} = \dfrac{n \cdot \binom{n-1}{m-1}}{m}$.
3. Finalització: *A cada crida recursiva, els dos arguments de la funció decreixen en una unitat. Això significa que en algun moment el segon argument tindrà el valor* 0*, cosa que finalitzarà les crides recursives.*

```
#include <iostream>
using namespace std;
int combinacions(int y, int z);
int main(void) {
int n, m;
cout << "Introduiu dos nombres: ";
cin >> n >> m;
cout << endl << "Les combinacions de " << n <<
" objectes agafats de " << m << " en " << m << " son ";
cout << combinacions (n,m) << endl << endl;
system("pause");
}
int combinacions(int n, int m) {
int r;
if (n==m || m==0) r=1; // casos base
else r=(n*combinacions(n-1,m-1))/m; // cas recursiu
return r;
}
```

Programa 57 *Conversió d'un nombre natural expresat en base* 10 *a base* 2*. En aquest cas, els tres apartats en què descomponem l'anàlisi de la funció recursiva són els següents:*

1. Cas base : *Si el nombre decimal a convertir és* 0*, el valor binari* $b(0)$ *és* 0*; si és* 1*, el valor binari* $b(1)$ *és* 1*. Això ho podem expressar també de la manera següent: si* $n < 2$*,* $b(n) = Resta(n/2)$*.*
2. Cas recursiu: *Si* $n > 2$*, per obtenir* $b(n)$ *concatenem el resultat de* $b(n/2)$ *amb el valor de* $Resta(n/2)$*.*
3. Finalització: *A cada crida recursiva, l'argument de la funció és la meitat del valor de la crida anterior. Això significa que en algun moment tindrà un valor més petit o igual a* 1*, cosa que finalitzarà les crides recursives.*

```
#include <iostream>
using namespace std;
void DaB(int n);
int main(void)
{
        int nombre;
        cout << "Introduiu un nombre a representar en binari: ";
        cin >> nombre;
        cout << "En binari: ";
        DaB(nombre);
        cout << endl;
        system ("pause");
}
void DaB(int n)
{
        int a,R;
        R = n % 2;
        if (n > 1){
           a = n / 2;
           DaB(a);
        }
        cout << R;
}
```

4.10 Exercicis proposats de recursivitat

Enunciat 1: Integral

Implementació d'un programa C++ complet, és a dir, que contingui definicions de constants, tipus, prototipus, funció *main* i funcions auxiliars. El programa ha de calcular el valor aproximat de l'àrea delimitada per l'eix x i el valor d'una funció contínua $f(x)$ entre dos punts $x = a$ i $x = b$. El valor real és el valor de la integral definida entre a i b. $\int_a^b f(x)$. El valor aproximat l'obtindrem dividint l'interval $[a, b]$ en N intervals més petits i considerant la suma de cadascun dels rectangles que tenen com a base el valor de l'interval i com a altura, el valor de la funció a l'inici de cada interval, segons es mostra a la figura 4.2.

Per simplificar, considerem funcions del tipus $f(x) = k \cdot x$. En aquest cas, la integral $F(x)$ és la funció $F(x) = k \cdot \frac{x^2}{2}$ i la integral definida entre els punts a i b, és a dir, l'àrea compresa entre la funció i els punts a i b, és el valor $F(b) - F(a)$.

El coeficient k, el nombre d'intervals N i els valors a i b es demanen per pantalla. A partir d'aquests valors, calculem la variable global $base = (b - a)/N$, que serà la base de cada rectangle.

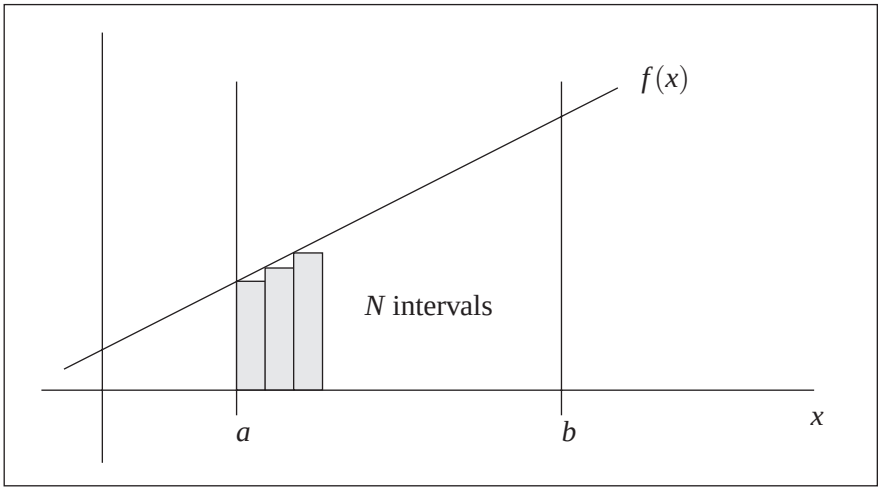


Fig. 4.2 Càlcul integral

El programa inclourà els apartats següents:

La funció double *funcio(double k, double x)*, que retorna el valor de la funció $f(x) = k \cdot x$. La funció double *Funcio(double k, double x)*, que retorna el valor de $F(x) = k \cdot \frac{x^2}{2}$, és a dir, el valor exacte de la integral.

La funció recursiva double *Integra(double k, double a, double b)*, que calcula el valor aproximat de l'àrea. Aquesta funció ha de finalitzar quan el valor de l'argument a sigui més gran que el valor de l'argument b o igual. A cada nova iteració, cal incrementar el valor de a amb el valor de la variable *base*.

Definiu una funció *int main(void)* que utilitzi les funcions anteriors i permeti:

- Demanar a l'usuari els valors de k, N, a i b. Per tal que l'usuari sàpiga en tot moment la variable que ha d'introduir, cal que es mostri un missatge que informi de la variable que s'hi ha d'introduir. Cal validar que $b > a$.
- Per finalitzar el programa, l'usuari ha d'entrar el valor 0 en el nombre d'intervals.
- Cal mostrar el valor real i aproximat de l'àrea calculada.

Enunciat 2: Vasos comunicants

Un sistema de vasos comunicants consisteix en uns dipòsits que contenen líquid a diferent altura i que estan comunicats per un tub. En aquestes condicions, el nivell d'un dels dipòsits baixa mentre que el de l'altre dipòsit puja fins a quedar al mateix nivell.

Simularem un sistema de dos dipòsits cilíndrics de radis R_1 i R_2, connectats mitjançant un tub. Suposant que $R_1 <= R_2$, que l'altura dels cilindres és h i que el primer d'ells ($radi = R_1$) s'omple completament de líquid, calculeu l'altura h' del líquid a la qual s'estabilitzarà el sistema. Suposem que podem despreciar el volum d'aigua que hi ha dins el tub de comunicació (figura 4.3). El volum d'un cilindre és $\pi \cdot R^2 \cdot h$.

Plantejem la solució de la manera següent: per cada decrement d'altura del nivell de líquid del cilindre de radi R_1 (Δh_1), s'incrementa el nivell del líquid del cilindre de radi R_2 en la quantitat Δh_2, essent $\Delta h_2 = \frac{R_1^2 \Delta h_1}{R_2^2}$. Anirem decrementant el nivell inicial del cilindre $R_1(h_1)$ en $0{,}001$ unitats fins que el nivell dels dos cilindres sigui igual. Admetem un marge d'error de $0{,}001$ unitats, és a dir, considerem que els nivells són iguals si $abs(h_1 - h_2) <= 0{,}001$.

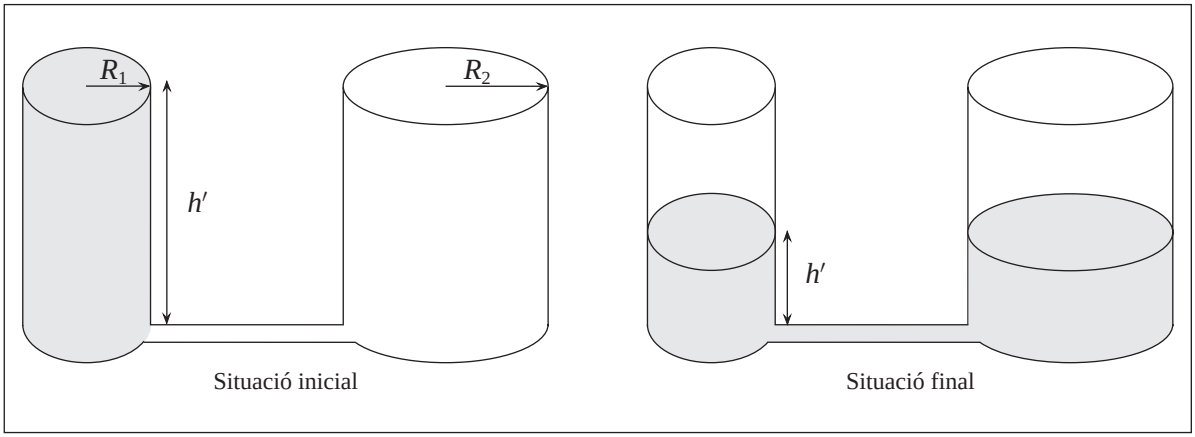


Fig. 4.3 Vasos comunicants

El programa ***main*** ha d'incloure:

- Demanar a l'usuari els valors de R_1, R_2 i h. Per tal que l'usuari sàpiga en tot moment la variable que ha d'introduir, cal que es mostri un missatge que informi de la variable que s'hi ha d'introduir. Cal validar que $R_1 <= R_2$.
- Una funció double ***increment(double h1, double R1, double R2)***, que calcula l'increment de nivell h_2 en funció del decrement h_1.
- Una funció recursiva double ***nivelar(double error, double h, double R1, double R2)***, que retorna el nivell en què s'estabilitza el sistema.

5 Taules

5.1 Introducció

Al capítol 2 d'aquest llibre, s'han explicat els conceptes fonamentals per programar en C++. Un d'ells és el concepte d'objecte, que fins ara només s'ha considerat com un lloc de la memòria on poder guardar un valor d'algun tipus bàsic. Però, a vegades, a l'hora de programar, ens podem trobar amb la necessitat d'haver de processar un conjunt de valors que estan relacionats entre si, per exemple: una llista de qualificacions, una taula de temperatures, etc. El processament d'aquests conjunts de dades utilitzant objectes simples, individuals, podria arribar a ser molt complex; per això, la majoria de llenguatges de programació disposen de regles sintàctiques per processar aquestes agrupacions de dades, definides o emmagatzemades en variables compostes o estructurades, entre les quals es troben les taules.

5.2 Concepte de taula

Suposem que s'ha de fer un programa que, donada una seqüència de 50 enters, escrigui per pantalla quin d'aquests enters es repeteix més vegades. Evidentment, amb el que s'ha vist fins ara en capítols anteriors, sembla que el nombre de variables a definir hauria de ser més de 50, almenys una per a cada enter. A tot això cal afegir la dificultat per guardar els valors a cada variable, en cas de voler utilitzar un *esquema de recorregut de seqüències*, ja que per resoldre el problema no es poden perdre els valors que es van llegint, i al final en resulta un codi molt poc interessant. Solució: utilitzar una taula. Però, què és una taula?

> Una taula és un objecte format per un nombre fixat d'elements del mateix tipus.

Una taula també es pot veure com un conjunt de variables del mateix tipus i independents, que comparteixen el nom o identificador i a les quals es pot accedir mitjançant un índex. Tal com succeeix amb les variables bàsiques, les taules són estàtiques i la seva grandària ha de ser coneguda en temps de compilació; d'aquesta manera, el compilador podrà reservar la memòria necessària.

> Una taula és el mecanisme més senzill d'agrupació de dades.

Una representació gràfica d'una taula de 10 elements podria ser el dibuix que hi ha a continuació, on cada tros representaria una variable d'algun tipus definit prèviament.

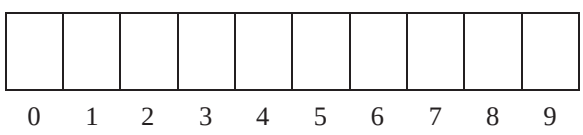


Fig. 5.1 Representació gràfica d'una taula

5.3 Taules d'una dimensió

En el món real, la dimensió d'un objecte és la mesura que en defineix la forma i la grandària. Les taules, com objectes que són, poden ser de diferents dimensions: d'una, de dues o matrius, de tres o tridimensionals, etc. En general, es parla de dos grups, les unidimensionals i les multidimensionals a partir dues, però en aquest llibre es tracten només les taules de fins a dues dimensions. Cada element d'una taula s'anomena *component* i té unes característiques determinades:

Les taules d'una dimensió també s'anomenen *vectors*.

- Es pot accedir i referenciar cada component aleatòriament.
- El nombre de components definits és inalterable.

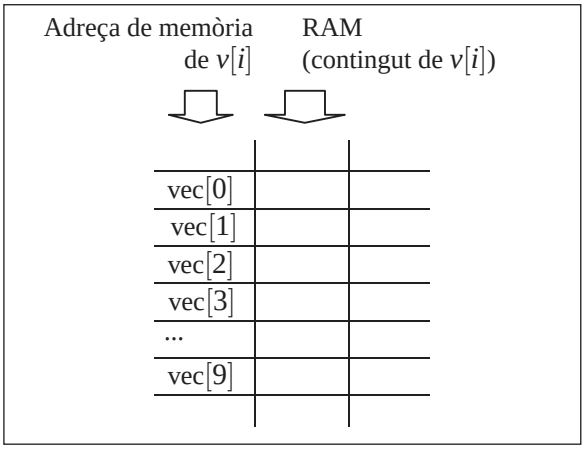


Fig. 5.2 Vector emmagatzemat a la RAM

- Per accedir als components, es necessita un índex, el qual prendrà valors dins d'un tipus ordinal, generalment dins els enters, i es pot calcular avaluant una expressió.
- Un component de la taula s'identifica amb el nom de la variable taula i l'índex entre [ ].
- Cada component de la taula és, en si mateix, una variable del tipus que s'hagi definit la taula; per tant, a cada component se li poden aplicar les operacions definides dins d'aquest tipus, igual com es faria amb una variable simple.
- La introducció d'informació en una taula s'ha de fer component a component.
- No existeix l'operació d'assignació entre taules; per això, a l'hora de copiar la informació d'una taula a una altra s'ha de copiar component a component.

La inicialització d'una variable taula en C++ no és automàtica. Per aquest motiu, les seves components tindran valors sense sentit fins que s'inicialitzi amb els valors adequats.

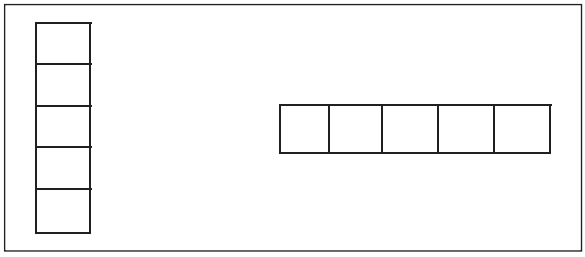


Fig. 5.3 Taula vertical/horitzontal

5.3.1 Definició com a variable

Una taula d'una dimensió, gràficament, es pot veure de dues maneres:

Declarar una taula d'una dimensió com si fos una variable és una de les dues maneres de declarar taules (a l'apartat següent hi ha explicada l'altra). La sintaxi de definició és molt semblant a l'emprada per declarar variables elementals; l'única diferència és que el nom de la variable, en aquest cas, ha d'anar acompanyat del nombre d'elements, tal com es mostra a continuació.

<TipusComponent> <NomTaula> [<NombreElements>];

Per exemple, *float m*[3];

- *<NombreElements>* ha de ser un enter i representa el nombre de components que tindrà la taula. Pot ser també el resultat d'avaluar una expressió entera o una constant, i ha d'anar obligatòriament entre els caràcters [ ].

- $<TipusComponent>$ ha de ser el nom d'un tipus elemental o definit prèviament. Un cop declarada la variable taula, igual que en el cas de les variables simples, es disposa d'un identificador (el nom de la taula) que referencia un conjunt de dades del mateix tipus. A partir d'aquest moment, s'hi pot guardar informació, extreure'n, etc.
- S'accedeix a les components de la taula unidimensional utilitzant l'operador [ ].
- En C++, la posició del primer element de la taula és la 0 i la de l'últim és la que ocupa la posició $<nombreElements> - 1$.

Igual que en el cas de les variables de tipus bàsic, les taules també es poden inicialitzar amb qualsevol valor del tipus de les components. La sintaxi que cal utilitzar en aquests casos és:

$$< TipusComponent >< Variable[nombreElements] >= \{valor_1,\ldots,valor_n\}$$

Per exemple: *int vec[4] = {1, –3, 100, 24};*

5.3.2 Accés

Hi ha una relació biunívoca (un a un) entre els valors que pren l'índex i les components de la taula, però cal anar amb molta cura a l'hora de treballar amb taules ja que, si no es té control dels rangs dels índexs, com que a la memòria es guarden totes les variables que s'han definit en el programa, si l'índex pren, per error, valors no desitjats com succeeix a l'exemple que hi ha a continuació, pot passar que les variables simples (en aquest cas, la b) canviïn de valor sense haver fet una assignació explícita, només canviant el contingut d'un component de la taula que està fora de rang de l'índex però que, com a posició relativa començant a comptar des de la 0, sí que existeix. Això pot ocasionar greus problemes, com es pot veure al programa següent:

> L'accés a una component d'aquest tipus de taules es fa mitjançant un índex i pot ser per escriure, llegir, esborrar o modificar-ne el contingut.
>
> La lectura o escriptura i, en general, qualsevol processament fet a totes les components d'una taula requereix sempre alguna estructura repetitiva *while* o *for*.

Programa 58

Exemple per il·lustrar la ubicació de les variables a la memòria.

```
#include <iostream>
using namespace std;
int main() {
const int N = 5;
int a[N] = {10,20,30,40,50};
int b = 9000;
cout<<"b = ";
cout<<b;
cout<<" i es troba a a[-1] ";
cout<<endl;
for ( int i = 0; i < 7; i++ ) {
    cout<<"a["<<i<<"]=";
    cout<<a[i];
    cout<<endl;
}
cout<<"Si posem a[-1]=51423";
```

```
cout<<"canviarà el valor de b";
cout<<endl;
a[-1] = 51423;
cout <<"Ara la variable b=";
cout<<b;
cout<<endl;
system("pause");
return 0;
}
```

Si s'executa, per pantalla mostrarà el contingut de cadascuna de les components de la taula a:

```
Valor d' a[0]=10
Valor d' a[1]=20
Valor d' a[2]=30
Valor d' a[3]=40
Valor d' a[4]=50
Valor d' a[5]=0
Valor d' a[6]=229364
```

> A diferència de les variables de tipus bàsic, en les variables taula no es pot utilitzar la sentència d'assignació.

Per exemple, si intentem compilar el programa que hi ha a continuació, donarà un error a la línia de codi en què es fa l'assignació d'y a $x\,(x = y)$.

```
//Programa d'assignació de taules

#include <iostream>
using namespace std;
const int N = 5;

int main() {
  int x[N] = {0, 1, 2, 3,  4};          //inicialització de la variable x
  int y[N] = {1, 2, 4, 8, 16};          //inicialització de la variable y
  x = y;                                 //assignació de la variable y a l'x
  return 0;
}
```

Programa 59

Dissenyar un programa que escrigui per pantalla quantes vegades es repeteix l'últim element d'un conjunt de 50 *enters.*

Passos que cal tenir en compte en el disseny:

1. Definició i inicialització de variables

   En aquest exemple, a banda de la taula per guardar cadascun dels 50 enters, necessitem dues variables més de tipus *enter*: una que serà l'índex de la taula i l'altra, per comptar les vegades que es repeteix

l'últim element. Com es pot observar, la taula *t[ELEMENTS]* es defineix en el lloc habitual destinat a la definició de variables.

A l'hora de dissenyar un programa, és recomanable que el nom de les variables ajudi a recordar la informació que emmagatzemen; per això, la variable taula l'anomenarem *t[ELEMENTS]*; l'índex, *índex*, i el comptador, *cont.* Com que la variable *cont*, cada vegada que es repeteixi l'últim element en la seqüència ha d'incrementar el seu valor en una unitat, inicialment se li donarà el valor 0.

2. Cos del programa

Llegir els elements de la seqüència i guardar-los a la taula.

L'enunciat diu que la seqüència tindrà 50 elements, per això no hi ha sentinella. Els elements es van guardant a la variable taula en l'ordre que es van introduint pel teclat, començant per la posició 0. L'última serà la 49 ja que de 0 a 49, ambdues incloses, hi ha 50 posicions. El plantejament d'aquesta part del problema consta alhora de dues parts:

1. Aplicar un esquema de recorregut als elements de la seqüència d'entrada a mesura que es van introduint a la taula.
2. Aplicar un segon esquema de recorregut als elements de la taula des de la posició 0 fins a la 48 per comparar cadascun d'ells amb el de la posició la 49.

3 Tractament final

Mostrar per pantalla el resultat, és a dir, el nombre de vegades que es repeteix l'últim element.

El programa resultant és el següent:

```
#include <iostream>
using namespace std;
const int ELEMENTS=50;
int main(void) {
  int t[ELEMENTS],cont=0;              //variable taula
  int index=0;                         //índex per la variable taula
  cin>>t[index];
  while(index<ELEMENTS){
     cin>>t[index];                    // omplir la taula
     index++;
  }
  index=0;
  while(index<ELEMENTS-1){
     if(t[ELEMENTS-1]==t[index]) cont++;
     index++;
  }
  cout<<"l'últim element es repeteix:¨<<cont<<" vegades";
  return 0;
 }
```

5.3.3 Definició com un tipus nou

Una altra possibilitat a l'hora de treballar amb taules és definir-les com un nou tipus de dades, en lloc de fer-ho com una variable, tal com s'ha explicat a l'apartat anterior. Però, si es vol definir com un nou tipus, evidentment cal que abans de la definició de la variable taula s'hagi definit aquest nou tipus.

La sintaxi emprada, en aquest cas, per a la definició:

$$typedef < tipusComponent >< nomNouTipus > [< nombreElements >]$$

typedef és una paraula reservada (també s'anomena directiva) de C++, en què una de les seves utilitats és assignar un àlias a un tipus ja existent, és a dir, crea un nou identificador de tipus per a un tipus que ja té el seu propi i que, a partir del moment que s'ha definit aquest identificador, es podran intercanviar lliurement en qualsevol expressió. La utilització de *typedef* a vegades pot facilitar molt la lectura dels programes, ja que permet escriure el codi amb paraules més senzilles o expressives. Per exemple, si us fixeu en el codi següent:

```
#include<iostream>
using namespace std;
int main(){
   double notes;
   notes=9.5;
   cout<<notes<<endl;
}
```

No fa altra cosa que escriure per pantalla el contingut de la variable *notes*, però el que hi ha a continuació

```
#include<iostream>
using namespace std;
typedef double notesAlumne;
int main(){
  notesAlumne notes=9.5;
  cout<<notes<<endl;
}
```

Fa el mateix. L'única diferència és que en aquest últim s'ha definit un nou tipus, com un àlies del tipus bàsic *double* per definir les notes. El resultat és el mateix, però sembla que el tipus *notesAlumne* és més idoni en aquest context.

També es podria definir una constant de forma semblant:

```
typedef const float REAL;
REAL pi = 3.14;
```

Com es pot definir una variable utilitzant aquest nou tipus? De la mateixa manera que en el cas de les variables de tipus bàsic, i la seva sintaxi és:

```
nomNouTipus nomDeLaVariable;
```

Al codi del programa 59 es pot veure que defineix la variable *t*, de la manera següent:

```
int t[ELEMENTS];
```

Ara, si es vol fer el mateix però definint un nou tipus per a la variable *t*, cal posar el següent:

```
const int ELEMENTS=50;
typedef int taula[ELEMENTS];
```

Si es vol definir una variable de tipus taula:

```
const int ELEMENTS=50;
typedef int taula[ELEMENTS];
int main(void) {
taula t;
```

L'avantatge, en aquest cas, és que es disposa d'un nou tipus anomenat *taula*, que pot guardar 50 enters i que, com en el cas dels tipus bàsics, es pot definir tantes variables d'aquest tipus com calgui. Per exemple:

```
taula t,v,w,u;
```

on les quatre seran taules d'enters de 50 components.

5.4 Taules de dues dimensions

A la secció 5.3 s'ha vist la teoria sobre taules d'una sola dimensió. Ara es veuran els mateixos conceptes, però per a taules de dues dimensions, les quals es poden entendre com una extensió de les primeres.

Des del punt de vista del programa, una taula de dues dimensions no és altre cosa que una zona d'emmagatzemament contigu (com en el cas d'una dimensió) on hi ha un conjunt d'elements del mateix tipus. Cal tenir present que, a l'hora de treballar amb aquests tipus de variables, si tenen més de tres dimensions, el disseny del codi per processar-les pot ser complicat. El més habitual és treballar amb taules de dues dimensions com a màxim.

Les taules de dues dimensions també s'anomenen *matrius*.

Des del punt de vista lògic, una taula de dues dimensions es pot considerar un conjunt d'elements ordenats per files.

5.4.1 Definició

La sintaxi de definició per al cas de N dimensions seria:

$$< tipus >< NomTaula > [< Dim_1 >] \cdots [< Dim_n >]$$

on $< Dim\_i >$ és un enter que representa el nombre de components que tindrà la *dimensió_i*, que, com en el cas unidimensional, pot ser el resultat d'avaluar una expressió entera o una constant. Tant el nombre d'elements de les files com el de les columnes han d'estar entre els caràcters [ ].

La sintaxi de definició per al cas de dues dimensions, seguint la nomenclatura del cas general, serà:

$$< tipus >< NomTaula > [< Dim_1 >][< Dim_2 >]$$

Un exemple d'aquest tipus de definició podria ser:

```
int matriu[3][3];
```

5.4.2 Accés

En aquest tipus de taules, s'utilitza un índex per a cada dimensió, és a dir, un per a les files i un altre per a les columnes. S'hi pot accedir per files o per columnes, de la mateixa manera que s'accedeix en matemàtiques a una matriu d'enters o reals; tot depèn de la necessitat del problema.

Per il·lustrar aquest apartat, s'utilitza com a exemple el càlcul de la suma de dues matrius, ambdues de 3×3 components, que són les que hi ha a continuació.

$$\begin{pmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \\ 1 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \\ 2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+0 & 2+5 \\ 1+7 & 0+5 & 0+0 \\ 1+2 & 2+1 & 2+1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \\ 3 & 3 & 3 \end{pmatrix}$$

Programa 60

Dissenyar un programa que, donada una matriu constant de 3×3 *la imprimeixi per pantalla.*

Passos que cal tenir en compte en el disseny:

1. Definició i inicialització de variables
2. Inicialitzar (omplir) les matrius amb valors constants

   En ser una matriu constant, els valors no s'introduiran des de teclat, sinó que s'inicialitzarà directament amb valors constants.
3. Mostrar la matriu per pantalla

   S'utilitzaran dues sentències *for*, una per les files (és la més externa) i l'altra per les columnes.

```
#include <iostream>
using namespace std;
int main (){
  int matriu[3][3];
  matriu[0][0]=2;          //inicialització de la matriu a valors constants
  matriu[0][1]=3;
  matriu[0][2]=4;
  matriu[1][0]=1;
  matriu[1][1]=2;
  matriu[1][2]=3;
  matriu[2][0]=1;
  matriu[2][1]=2;
  matriu[2][2]=4;
   for (int i=0; i<3; i++)  //imprimir la matriu per pantalla
     for (int j=0; j<3; j++)
        cout<<"matriu["<<i<<","<<j<<"]= ''<<matriu[i][j]<<endl;
 return 0;
}
```

Programa 61

Dissenyar un programa que llegeixi una matriu d'enters de 10×10 *elements, sumi els valors de cadascuna de les files i guardi el resultat en un vector suma.*

Passos que cal tenir en compte en el disseny:

1. **Definició de les variables necessàries**

   Una taula de dues dimensions i de 10×10 components, és a dir, de 10 files per 10 columnes d'enters. Per definir la taula, hi ha dues possibilitats:

   1. Una possible definició pot ser:

      typedef int matriu[10][10];//tipus matriu
      typedef int vector[10]; //tipus vector pel vector suma
      matriu M; //variable matriu
      vector V; //variable vector

   2. Una altra seria definir directament les variables:

      int M[10][10]; //variable matriu
      int V[10]; //variable vector pel vector suma

2. **Omplir la matriu**

   Es pot utilitzar un *while* o un *for*, indistintament. En aquest cas, s'utilitzarà el *for* i la solució del problema es pot plantejar de dues maneres:

   1. Omplir la matriu primer i després recórrer-la una altra vegada per sumar els elements de cada fila.
   2. A mesura que es va omplint la matriu, anar sumant els elements de cada fila i guardar el resultat de la suma a la posició corresponent del vector suma.

   En ambdós casos, la suma de la *fila i* de la matriu es col·locarà a la posició *V[i]*. S'ha optat per la segona opció perquè és millor que la primera a l'hora d'optimitzar el temps d'execució del programa, ja que per inserir i sumar els elements de la matriu, només s'ha de recórrer una vegada i, evidentment, això és estalviar temps que, encara que no sigui l'objectiu primordial d'aquest llibre, no deixa de ser important. S'ha de dir també que, atès que el contingut de *V[i]* s'ha d'acumular a cada element de la fila de la matriu en cada passada, aquest vector s'ha inicialitzat a zeros de bon principi. És cert que es podia fer en un altre punt del programa, però s'ha fet aprofitant la definició. Queda per al lector esbrinar on es podia haver posat sense que n'alterés el resultat final.

3. **Mostrar la informació introduïda per pantalla**

   Evidentment, seguint el mateix raonament que en el pas anterior, mentre es calcula la suma, es pot imprimir alhora el resultat per pantalla, és a dir, el contingut del vector *V*, però, tal com es pot observar, no s'ha fet. Es deixa per al lector com a opció de millora del programa.

```
#include <iostream>
using namespace std;
const int FILES = 10;
const int COLUMNES = 10;
int main() {
int matriu[FILES][COLUMNES], i,j;
int vector[FILES]={0,0,0,0,0,0,0,0,0,0};
for(i=0; i<FILES; i++)
  for(j=0; j<COLUMNES; j++) {
```

```
        cout<<"Element "<< i<<","<<j<<" de la matriu: ";
        cin>>matriu[i][j];
        vector[i]=vector[i]+matriu[i][j];
  }
cout<<"La matriu generada és:"<<endl;
for(i=0; i<FILES; i++){
  for(j=0; j<COLUMNES; j++){
     cout<<matriu[i][j];
  }
    cout<<endl;
}
cout<<"I el vector suma de las files és: "<<endl;
for(i=0; i<FILES; i++)
  cout<<vector[i]<<endl;
return 0;
}
```

5.5 Taules amb component taula

La definició de taula de la secció 5.2 diu que les components han de ser totes del mateix tipus, però no diu de quin, per això poden ser perfectament un tipus taula.

> Les taules amb components taula són aquelles que el tipus de les seves components són d'algun tipus taula definit prèviament.

Ho veurem mitjançant un exemple:

Programa 62

Dissenyar un programa per guardar les notes introduïdes des de teclat d'una classe de 100 *alumnes on cada alumne està matriculat en cinc assignatures.*

Passos que cal tenir en compte en el disseny:

1. **Definició dels tipus i variables i com accedir a la informació**

   En principi, una taula de 100 posicions, en una posició qualsevol guardarà les notes que hagi tingut un alumne en cadascuna de les cinc assignatures en què estigui matriculat. Igual que en la definició de variables de l'enunciat 5.4.2, es pot fer de dues maneres:

   a) *typedef double notes[5];* // tipus notes
   typedef notes taulaNotes[100]; // tipus alumnes
   taulaNotes alumnes; // variable alumnes

   b) *typedef double notes[5];* // tipus notes
   notes alumnes[100]; // variable alumnes

   - La primera nota del primer alumne, considerant la variable ***alumnes*** procedent de qualsevol de les dues definicions anteriors, seria la que està guardada a la posició ***alumnes[0][0]***, és a dir, ***alumnes[0]*** seria la posició de la taula que conté totes les notes de l'alumne que ocupa la posició 0 i, com que s'ha dissenyat una taula de taules, dintre d'aquesta posició hi ha una altra taula que

té les notes de les cinc assignatures d'aquest alumne; per tant, la primera nota serà la de la posició 0 d'aquesta subtaula (*alumne[0]*), l'element *alumnes[0][0]*.

Aquesta notació no s'ha de confondre amb la que s'utilitza per a l'accés a les taules de dues dimensions. En aquest cas, succeeix que cada posició o component de la taula *alumnes[i]* és alhora una altra taula, cosa que conceptualment no seria així si s'hagués definit una matriu. Ara, seguint la lògica de la definició, només cal utilitzar un altre índex i, d'aquesta manera, la nota *j* de l'alumne *i* es podria representar com mostra la figura 5.4.

En general, el processament de les taules requereix la utilització d'un *while* o un *for* per a cada dimensió.

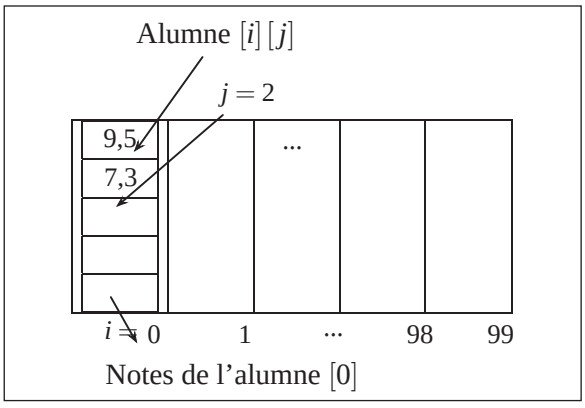


Fig. 5.4 Taula amb components taula

2. Omplir la taula

Es fa com en el cas de les matrius, utilitzant dues sentències repetitives, seguint el criteri que s'ha exposat a la secció 5.4. Per a aquest problema, s'han utilitzat dos *for*. El primer permetrà recórrer la taula d'alumnes utilitzant l'índex *i*, i el segon per guardar cadascuna de les notes d'un alumne. En el segon *for*, només cal posar dues instruccions: una per informar l'usuari de les dades que ha d'introduir i l'altra per recollir el valor introduït.

3. Mostrar la informació introduïda per pantalla

Ara només resta mostrar per pantalla la informació que es demana a l'enunciat, cosa que es fa també amb dues instrusions *for*, ja que s'ha de recórrer una altra vegada la taula, amb la diferència que aquesta vegada, en lloc de guardar la informació (*cin*), s'ha de mostrar per pantalla (*cout*) la que s'hi ha guardat en el pas 2.

```
#include <iostream>
using namespace std;
const int ASSIGNATURES = 5;
const int ALUMNES = 100;
typedef double notes[ASSIGNATURES];        // tipus notes
typedef notes taulaNotes[ALUMNES];         // tipus alumnes
int main(){
taulaNotes alumnes;                        // variable alumnes
int i,j;
for(i=0; i<ALUMNES; i++)
  for(j=0; j<ASSIGNATURES; j++){
     cout<<"Nota de l'assignatura "<< j <<" de l'alumne "<<i <<" : ";
     cin>>alumnes[i][j];
  }
for(i=0; i<ALUMNES; i++){
  for(j=0; j<ASSIGNATURES; j++){
     cout<<"La nota de l'assignatura "<< j <<" de l'alumne "<<i <<" és: ";
     cout<<alumnes[i][j];
  }
}
return 0;
}
```

5.6 Taules en els paràmetres formals

Igual que les variables bàsiques, les taules i, en general, totes les variables *compostes* o *estructurades*, també poden formar part dels paràmetres formals. Per això, només cal remarcar una cosa: encara que el paràmetre de tipus taula no vagi acompanyat de &, el pas és per referència, ja que el seu identificador (el nom) conté l'adreça de memòria on comença la taula dins de la memòria de la màquina.

> Si es vol que una taula faci el pas de paràmetres per valor, s'ha d'utilitzar la directiva *const*.

Per als paràmetres formals de tipus taula, cal tenir en compte:

- Quan es fa la crida, no reserven espai dins la memòria per a la taula.
- No és necessari indicar el nombre d'elements que té la taula, és a dir, si la línia de capçalera és:

  void accioA(int M[10]), és equivalent a

  void accioA(int M[])
- L'adreça de memòria on està ubicada la taula és coneguda (es passa) en el moment de la crida de l'acció o funció.

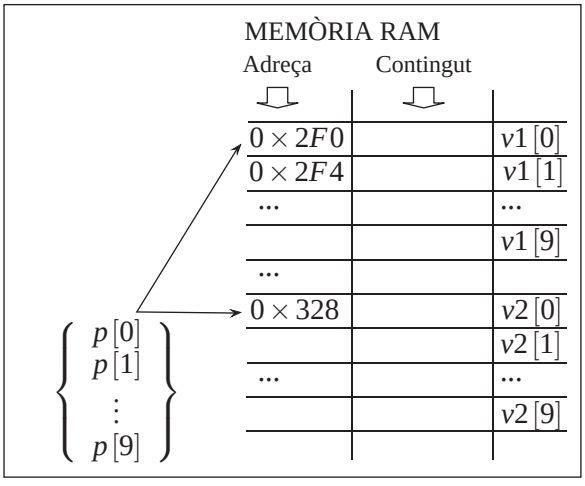


Fig. 5.5 Vector dins la RAM

Exemple

```
#include<iostream>
using namespace std;

const int ELEMENTS = 10;

void inicialitzar( float p[] ) {
int i;
for (i=0; i<ELEMENTS; i++)
  p[i] = 0;
}

int main() {
 float v1[ELEMENTS], v2[15];
 inicialitzar(v1);
 inicialitzar(v2);
 return 0;
}
```

El paràmetre p recull l'adreça de memòria on estan ubicades tant la taula $v1$ com la $v2$. No importa que siguin de diferent grandària, ja que realment només guarda l'adreça on comença cadascuna, que és la de la posició 0.

5.7 Esquemes de tractament de seqüències aplicats a taules

Es pot associar el contingut d'una taula unidimensional t, de tamany N, a una seqüència de N elements. Per exemple, una taula d'enters guardats en ordre creixent de l'índex es podria tractar com una seqüència considerant:

- Primer element: el $t[0]$.
- Element k: el $t[k-1]$.
- Darrer element: el $t[N-1]$.

Sempre es poden aplicar els esquemes de tractament de seqüències en algun sentit de l'índex, ja sigui creixent o decreixent, per exemple:

Programa 63

Dissenyar un programa que, donada una taula d'enters de 10 *elements, permeti saber quins són els* 3 *elements consecutius la suma dels quals és màxima. El programa ha d'escriure per pantalla la suma i els índexs dels elements en qüestió.*

Passos que cal tenir en compte en el disseny:

1. **Definició dels tipus i les variables necessàries**

   Es necessita una taula d'enters de 10 components, un comptador per comptar les posicions de tres en tres, una variable de tipus enter per guardar les sumes parcials dels elements i una variable entera per guardar l'índex de l'últim element dels tres candidats, la suma dels quals sigui la màxima, per poder imprimir per pantalla les seves posicions.

2. **Omplir la taula des del teclat i estudiar-ne els elements de la manera següent**

   Començant per la posició 0 de la taula, sumar el contingut de les tres primeres posicions. Un cop sumades, posicionar l'índex i a la segona posició $(v[1])$ i sumar-la amb les dues següents, comprovar si la suma és més gran que l'anterior i posicionar l'índex a $v[2]$ i, així successivament fins que l'índex arribi a valer $ELEMENTS-1$, ja que en aquest moment no cal incrementar-lo més, perquè ja no queden tres elements per sumar a la taula. En el codi del programa que hi ha a continuació, tot això es controla en l'últim *if*:

   if(i<ELEMENTS-2) i=i-1; també es podia haver posat

   if(i<=ELEMENTS-1) i=i-1;

   ja que, mentre la i no arribi a valer 9, s'ha de decrementar una unitat cada vegada; per exemple, si en l'última passada s'han sumat les components $v[6]$, $v[7]$ i $v[8]$, la volta següent dins el *while* s'hauran de sumar les components $v[7]$, $v[8]$ i $v[9]$, és a dir, quan s'acaba de fer la suma, l'índex i val 8, però per a la suma següent s'ha de decrementar en 1. Això passa fins que la i val 9, que no cal fer res, perquè no queden tres elements per poder sumar.

3. **Mostrar el resultat per pantalla**

   Només cal imprimir per pantalla el valor de les variables *indexMax-2*, *indexMax-1* i *indexMax*, juntament amb el contingut de la variable *max*, que és on hi ha la suma.

```
#include <iostream>
using namespace std;
const int ELEMENTS = 10; //dimensió de la taula
void omplirTaula(int p[]);
int main() {
  int v[ELEMENTS], suma=0;
  int i=0,max=0,cont=1, indexMax;
  omplirTaula(v);
  while(i<ELEMENTS){
     suma=suma+v[i];
```

```
      cont++;
      if (cont==3){        //aquest if fa les sumes dels tres elements consecutius
         if(max<suma){
            max=suma;      //max conté la suma màxima
            indexMax=i;    //indexMax és l'índex de l'últim dels tres elements
            cont=0;
            suma=0;        //inicialitzar variables per estudiar els tres
          }                //següents elements
         if(i<ELEMENTS-2) i=i-1;
      }
      else i++;            //incrementa l'índex fins a tres consecutives
   }
   cout<<"els tres elements consecutius que sumen mes son";
   cout<<indexMax-2<<", "<<indexMax-1<<" i"<<indexMax;
   cout<<" i la seva suma es: "<< max;
   system("pause");
   return 0;
}
void omplirTaula(int p[]) {
int i;
for (i=0; i<ELEMENTS; i++)
   cin>>p[i];
}
```

5.7.1 Esquema de recorregut aplicat a una taula de dues dimensions

A la secció anterior, s'explica com aplicar un esquema de tractament de seqüències a una taula d'una dimensió. Doncs bé, seguint un criteri similar es pot aplicar a taules de dues, tres i fins a *n* dimensions, si és necessari.

Per al cas de taules de dues dimensions, les matrius, aplicar un esquema de recorregut és pràcticament el mateix que omplir-les, en el sentit que només cal utilitzar sentències repetitives per processar-les per files i dintre de cada fila processar totes les columnes per donar solució al problema, com es mostra a l'exemple que hi ha a continuació.

Programa 64

Dissenyar un programa que llegeixi una matriu d'enters de 3×3 *elements i escrigui per pantalla el nombre de parells i el de senars entre els seus elements.*

Passos que cal tenir en compte en el disseny:

1. Definició dels tipus i les variables necessàries

   Per a aquest problema, només se'n necessiten dues: una matriu de 3×3 components i un comptador per comptar la quantitat de nombres parells, *comptParells*. La quantitat de senars l'obtindrem al final fent la resta entre el nombre total d'elements de la matriu i *comptParells*.

2. Omplir la taula

   Els enters s'introduiran des de teclat i, un cop la taula plena, es farà un recorregut per files i, dintre de cada fila, per columnes (aquests recorreguts els fan els dos últims *for*) per tal d'analitzar-ne els elements i comptar quants parells hi ha:

if(matriu[i][j]%2==0) comptParells ++;

3. Mostrar el resultat per pantalla

   El valor de *comptParells i FILES*COLUMNES-comptParells*

```
#include <iostream>
using namespace std;
const int FILES = 3;
const int COLUMNES = 3;
int main(){
int matriu[FILES][COLUMNES], i,j;
int comptParells=0;
for(i=0; i<FILES; i++)
  for(j=0; j<COLUMNES; j++){
     cout<<"Element "<< i<<","<<j<<" de la matriu: ";
     cin>>matriu[i][j];
}
for(i=0; i<FILES; i++){
   for(j=0; j<COLUMNES; j++){
      if(matriu[i][j]%2==0) comptParells ++;
  }
}
cout<<"El nombre de elements parells és: "<<comptParells<<endl;
cout<<"El nombre de elements senars és: "<<FILES*COLUMNES-comptParells<<endl;
return 0;
}
```

5.7.2 Esquema de cerca aplicat a una taula de dues dimensions

La idea inicial és la mateixa que per al cas de recorregut explicat a l'apartat anterior, és a dir, es disposa d'una matriu a la qual s'haurà d'aplicar un esquema de cerca per esbrinar si hi ha algun element que compleix una determinada propietat, com a l'exemple que hi ha a continuació.

Programa 65

Dissenyar un programa que llegeixi una matriu d'enters de 3×3 *elements i escrigui per pantalla si entre els seus elements hi ha algun quadrat perfecte.*

Primer de tot, per resoldre el problema s'ha de saber que, en matemàtiques, un nombre és un quadrat perfecte si la seva arrel quadrada és un enter. Per exemple, el 4 ho és ja que la seva arrel quadrada és 2.

Passos que cal tenir en compte en el disseny:

1. Definició d'una taula de 3×3 posicions
2. Omplir la taula des del teclat i estudiar-ne els elements per veure si algun d'ells és un quadrat perfecte. Es pot fer de la manera següent:

   if(int(sqrt(double(matriu[i][j])))==sqrt(double(matriu[i][j])))

   trobat=true;

   Per posar aquesta condició de l'*if*, es necessita una llibreria de C++ que tingui l'arrel quadrada (*sqrt*) i convertir l'element de la matriu a *double*, perquè la funció *sqrt* està definida amb el paràmetre d'aquest tipus; si no, dóna error de compilació.

Per saber si l'element és un quadrat perfecte, el càlcul que s'ha de fer és comparar si la part sencera de l'arrel quadrada de l'element és igual a l'arrel quadrada de l'element. Si això succeix, es pot afirmar que l'element és un quadrat perfecte.

Un cop es té clar el càlcul, s'ha de fer la cerca entre els elements de la matriu per si n'hi ha algun, i aleshores, ja s'haurà solucionat el problema. Això suposa fer una cerca per files i una altra per columnes; d'aquí ve que la variable booleana *trobat* s'hagi de posar en els dos *for*. Si s'observa el codi, la cerca s'ha codificat amb la instrucció repetitiva *for* en lloc de la tradicional *while*, posant juntes les dues condicions que s'han de complir per seguir estudiant elements:

i<FILES &&!trobat;

j<COLUMNES && !trobat;

3. Mostrar el resultat per pantalla

```
#include <iostream>
#include <cmath>
using namespace std;
const int FILES = 3;
const int COLUMNES = 3;
int main(){
int matriu[FILES][COLUMNES], i,j;
int contParells=0;
bool trobat=false;
for(i=0; i<FILES; i++)
  for(j=0; j<COLUMNES; j++){
     cout<<"Element "<< i<<","<<j<<" de la matriu: ";
     cin>>matriu[i][j];
  }
for(i=0; i<FILES && !trobat; i++){
  for(j=0; j<COLUMNES && !trobat; j++){
    if(int(sqrt(double(matriu[i][j])))==sqrt(double(matriu[i][j]))) trobat=true;
  }
}
if(trobat){
  cout<<"Hi ha almenys un quadrat perfecte i és el de ";
  cout<<"matriu["<<i-1<<"]["<<j-1<<"]"<<endl;
  cout<<matriu[i-1][j-1]<<endl;
}
else cout<<"No n'hi ha cap"<<endl;
return 0;
}
```

5.8 Taules parcialment plenes

Suposeu que, a l'hora de fer un programa, en lloc de conèixer de bon principi la dimensió de la taula, tal com succeeix en els exemples que s'han vist fins ara, només se sap que s'ha de processar un conjunt de dades, *n* dades, i que quan s'acabin es rebrà un sentinella per avisar que no se n'introduiran més. Evidentment, el problema s'ha de resoldre, però, de quina grandària haurà de ser la taula?

Davant d'una situació com aquesta, sempre s'ha de suposar quantes dades es poden haver de guardar, en el cas pitjor, ja que està clar que un programa ha de funcionar per al cas més general, encara que en alguns casos la memòria reservada per les dades o, el que és el mateix, la grandària de la taula, sigui del

tamany que sigui, no s'ompli del tot. Com es resolen aquests casos?: Controlant el nombre de posicions ocupades que hi ha a la taula, i aquest control es pot fer de dues maneres:

1. Per longitud
2. Per sentinella

En el primer cas, cal saber el nombre de posicions plenes que hi ha a la taula. En el segon, s'utilitza un valor especial, sempre pertanyent al tipus que tenen els components de la taula, per indicar quin és l'últim element de la taula que s'ha de tractar. A continuació, es pot trobar un exemple aplicat a cada cas.

5.8.1 Control per longitud

Programa 66

Dissenyar un programa que, donada una seqüència d'enters positius acabada en -1 *que representa la tirada d'un dau, escrigui per pantalla quin és el número que surt més vegades. El dau, com a màxim, es tirarà* 100 *vegades.*

Passos que cal tenir en compte en el disseny:

1. Definició i inicialització de les variables necessàries

   Una taula per guardar els 100 enters en el cas pitjor, la *taula t*, una de tipus *enter*, *index*, que serà l'índex d'aquesta taula i que servirà per anar apuntant l'element que s'ha d'estudiar a cada moment; *index1* serà l'altre índex, que cada vegada recorrerà la taula sencera per veure quantes vegades es repeteix l'element apuntat per *index*; *cont* per comptar les vegades que es repeteix l'element apuntat per *index*; *maxCont* per guardar el nombre de vegades que es repeteix *maxNum*, que serà l'element que apareix més vegades a la taula, i, finalment, la variable *esRepeteix*, que conté l'element de la taula per estudiar, és a dir l'apuntat per *index*, que no és sinó *t[index]* i candidat a ser el que es repeteix més vegades en cada passada. La definició de la variable *esRepeteix* no és necessària, però s'ha definit pensant que aportaria claredat a l'hora d'interpretar el codi. Com es pot observar, una vegada més, la variable *t[ELEMENTS]* es defineix en el lloc on habitualment es defineixen les variables.

2. Llegir els elements de la seqüència i guardar-los a la taula

   No se sap quants enters s'introduiran; és per això que a la seqüència d'entrada hi ha sentinella. En aquest exemple, com en els casos anteriors, els elements es van guardant directament dins la variable taula segons l'ordre d'entrada començant per la posició 0, fins que es detecti el sentinella.

   En principi, no se sap quantes posicions ocuparan les dades, però segur que no superaran les 100 ja que aquesta dada està garantida per l'enunciat. El que interessa saber és el nombre de posicions ocupades que tindrà la taula, i aquesta informació es guarda a la variable *nombrePosicionsPlenes*, la qual permetrà optimitzar el temps d'execució ja que, a partir de la posició *nombrePosicionsPlenes+1*, se sap que no hi ha informació rellevant; per tant, no cal processar aquestes components, amb la qual cosa el programa estalviarà temps.

3. Processar els elements de la taula

   Es tracta de començar per la posició 0 i comptar quantes vegades es repeteix cada element, és a dir, el que hi ha a la posició *t[index]*. El mètode que s'emprarà serà esbrinar quantes vegades es repeteix un enter (el que hi ha a *t[index]*) dins una seqüència de *nombrePosicionsPlenes* elements, cada vegada que s'executi el programa i sempre amb *nombrePosicionsPlenes* més petit o igual que 100.

   Cal observar que amb una sola taula es resol el problema, això si, amb dos índexs, un per apuntar l'element a estudiar i l'altre per recórrer la taula per veure si es repeteix.

4. Mostrar per pantalla el resultat

Això implica imprimir per pantalla l'enter que es repeteix més vegades i quantes són, que no és sinó el contingut de les variables *maxNum* i *maxCont-1*. El -1 és perquè al programa, cada element es compara també amb si mateix, i això no es considera una repetició.

El programa resultant és el següent:

```
#include <iostream>
using namespace std;
const int ELEMENTS=100;
int main(void){
  int t[ELEMENTS];                          //variable taula
  int cont,maxCont=0,maxNum;
  int index=0, esRepeteix, index1;          //índex per la variable taula
  int nombrePosicionsPlenes;
  cin>>t[index];
  while(t[index]!= -1 && index<ELEMENTS){
     index++;                               //omplir
     cin>>t[index];
}
nombrePosicionsPlenes=index;                //guardem el nombre de posicions
while(index<nombrePosicionsPlenes){         //plenes
  esRepeteix=t[index];
  index1=0;
  cont=0;
  while(index1<nombrePosicionsPlenes){      // comparar elements
     if(t[index1]==esRepeteix) cont++;
        index1++;
  }
  if (cont>maxCont){
     maxCont=cont;
     maxNum=esRepeteix;
  }
  index++;
  }
  cout<<"el nombre que mes es repeteix és: "<<maxNum;
  cout<<" i es repeteix  "<<maxCont-1<<" vegades";
  return 0;
}
```

5.8.2 Control per sentinella

L'enunciat que hi ha a continuació és el mateix que l'anterior, l'única diferència és en la solució. En aquest cas, tal com diu el títol d'aquesta secció, el control del nombre de posicions ocupades de la taula és per *sentinella*. Per tant, el programa serà una mica diferent.

Programa 67

Dissenyar un programa que, donada una seqüència d'enters positius acabada en -1 *que representa la tirada d'un dau, escrigui per pantalla quin és el número que surt més vegades. El dau, com a màxim, es tirarà* 100 *vegades.*

Passos que cal tenir en compte en el disseny:

1. Definició i inicialització de les variables necessàries

   Una taula, per guardar els 100 enters en el cas pitjor, la taula *t*; una altra que serà l'índex d'aquesta taula per anar apuntant l'element que s'ha d'estudiar a cada moment; *index1*, l'altre índex que cada vegada recorrerà la taula sencera per veure quantes vegades es repeteix l'element apuntat per *index*; *cont* per comptar les vegades que es repeteix l'element apuntat per *index*; *maxCont* per guardar el nombre de vegades que es repeteix *maxNum*, l'element que apareix més vegades a la taula, i, finalment, la variable *esRepeteix*, que conté l'element de la taula per estudiar, és a dir l'apuntat per *index*, que no és sinó *t[index]* i candidat a ser el que es repeteix més vegades en cada passada. La definició de la variable *esRepeteix* no és necessària, però s'ha definit pensant que aportaria claredat a l'hora d'interpretar el codi. Com es pot observar, aquesta vegada també la variable *t[ELEMENTS]* es defineix en el lloc on habitualment es defineixen de variables.

2. Llegir els elements de la seqüència i guardar-los a la taula

   En aquest exemple, no se sap quants enters tindrà la seqüència; és per això que hi ha sentinella per indicar quan s'acaben els elements. Com en el cas anterior, els elements es van guardant directament dins la variable taula segons l'ordre d'arribada, començant en la posició 0 fins que es detecti el sentinella. Al principi, no se sap quantes posicions ocuparan les dades, però sí que és segur que no superaran els 100 elements segons diu l'enunciat. En aquest cas, interessa recórrer la taula fins que es trobi el sentinella que, igual que en el cas de control per nombre de posicions plenes, permetrà optimitzar el temps d'execució, ja que a partir de la posició anterior a l'ocupada pel sentinella se sap que no hi ha informació rellevant; per tant, no cal processar aquestes posicions de la taula i el programa estalviarà temps.

3. Processar els elements de la taula

   Es tracta de començar per la posició 0 i comptar quantes vegades es repeteix cada element, el que hi ha a la posició *t[index]*. El mètode que s'emprarà per a cada element serà esbrinar quantes vegades es repeteix un enter (el que hi ha a *t[index]*) dins una seqüència d'elements acabada en -1.

   Cal observar que amb una sola taula es resol el problema, això sí, amb dos índexs, un per apuntar l'element a estudiar i l'altre per recórrer la taula per saber si es repeteix.

4. Mostrar per pantalla el resultat

   Això implica imprimir per pantalla l'enter que es repeteix més vegades i quantes són, és a dir, el contingut de les variables *maxCont-1* i *maxNum*.

   Els quatre passos estan implementats en el programa següent.

```
#include <iostream>
using namespace std;
const int ELEMENTS=100;
int main(void){
int t[ELEMENTS];                          //variable taula
int cont,maxCont=0,maxNum;
int index=0, esRepeteix, index1;          //índex per la variable taula
cin>>t[index];
while(t[index]!= -1 && index<ELEMENTS){
  index++;                                //omplir
  cin>>t[index];
}
index=0;
while(t[index]!=-1){
```

```
  esRepeteix=t[index];
  index1=0;
  cont=0;
  while(t[index1]!=-1){                          // comparar elements
     if(t[index1]==esRepeteix) cont++;
        index1++;
  }
  if (cont>maxCont){
     maxCont=cont;
     maxNum=esRepeteix;
  }
  index++;
  }
  cout<<"el nombre que mes es repeteix és: "<<maxNum;
  cout<<" i es repeteix  "<<maxCont-1<<" vegades";
  return 0;
}
```

5.8.3 Longitud vs. sentinella

En les dues seccions anteriors, s'ha explicat com treballar amb taules parcialment plenes. S'ha vist que el control del nombre de posicions plenes d'una taula es podia dur a terme tenint en compte la longitud o mitjançant un sentinella, però, quina de les dues és més recomanable? Depèn del problema: si s'està tractant una seqüència d'elements que té el seu propi sentinella, està clar que es pot aprofitar aquest element i guardar-lo a la taula; d'aquesta manera, quan calgui processar-la serà d'utilitat. Pot donar-se també el cas d'haver de treballar amb una seqüència d'elements, en què en lloc de tenir sentinella, es conegui el nombre d'elements que té. En aquest cas, pot ser més útil controlar el nombre de posicions plenes (per longitud) que haver d'inventar un sentinella.

5.9 Insercció i esborrament de components en una taula

5.9.1 Inserció

1	3	7	−1						

Fig. 5.6 Taula d'enters

Suposeu que teniu una taula d'enters ordenada de forma estrictament creixent, com la que hi ha a la figura de l'esquerra.

Si s'hi vol inserir un 6, està clar que el 6 s'ha de col·locar a la posició del 7, el 7 a la del -1 i el -1 ha d'anar al final una posició més a la dreta de la que ocupa ara. Com es fa això? El problema diu que els elements de la taula estan ordenats de forma estrictament creixent; per tant, només s'ha de saber on s'ha d'inserir el nou element, la qual cosa es fa en el segon *while* del programa que hi ha continuació.

Programa 68

```
#include <iostream>
using namespace std;
const int ELEMENTS=100;
void inserirElements(int t[],int& index){
  index=0;
  cin>>t[index];
  while(t[index]!= -1 && index<ELEMENTS){
```

```
    index++;
    cin>>t[index];
  }
}
int main(void){
int t[ELEMENTS],nouElement;
int posicionsPlenes;
int index,inserir;
cout<<"Inserir elements: ''<<endl;
inserirElements(t,index);                    //inserir elements de la seqüència inicial
posicionsPlenes=index;
//inserint elements
cout<<"Inserir un element? 0=no/1=si''<<endl;
cin>>inserir;
while(inserir==1&& index<ELEMENTS){
  cout<<"Element a inserir ? '';
  cin>>nouElement;
  index=0;
  while(index < posicionsPlenes){
     if(t[index] < nouElement && t[index+1] > nouElement){//posició nou element
        int i = posicionsPlenes;
        while( i > index){                                  // inserir l'element
           t[i+1]= t[i];
           i--;
        }
        t[i+1]=nouElement;
        posicionsPlenes++;
     }
     index++;
  }
  cout<<"Inserir un element? 0=no/1=si''<<endl;
  cin>>inserir;
}
index=0;
if(index >= ELEMENTS) cout<<"no hi caben més elements'';
cout<<" El contingut de la taula és:''<<endl;
while(t[index]!= -1){
  cout<<t[index]<<" '';
  index++;
}
  cout<<endl;;
  return 0;
```

Sobre el codi no hi ha gaire per comentar, és un codi senzill. S'ha definit una acció per inserir els elements de la seqüència, i el paràmetre formal de la taula és *int t[]*, equivalent a posar *int t[ELMENTS]*, tal com s'ha explicat ja a la secció 5.6 d'aquest capítol. El programa permet inserir tants elements com es vulgui mentre l'índex no superi la grandària de la taula.

5.9.2 Esborrament

En la inserció es movien els elements de la dreta de la posició on s'havia d'inserir l'element una posició cap a la dreta, per fer lloc al nou element. En aquest cas, s'han de moure tots els elements que hi ha a la dreta

de l'element que es vol esborrar una posició cap a l'esquerra; d'aquesta manera, si l'element a esborrar és el *t[i]*, es col·loca el *t[i+1]* a la posició de *t[i]* i aquest quedarà eliminat automàticament. Evidentment, això suposa que s'ha d'actualitzar el nombre de posicions plenes que tindrà ara la taula perquè, si s'ha de tornar a processar, cal tenir cura de no revassar el tamany. El programa que hi ha a continuació detalla tot aquest procés.

> Esborrar un component d'una taula es pot considerar pràcticament l'operació inversa de la inserció.

```
#include <iostream>
using namespace std;
const int ELEMENTS=10;
void inserirElements(int t[],int& index){
  index=0;
  cin>>t[index];
  while(t[index]!= -1 && index<ELEMENTS){
    index++;
    cin>>t[index];
  }
}
void mostrarElements(int t[]){
  int index=0;
  while(t[index]!= -1){
     cout<<t[index]<<" ";
     index++;
  }
}
int main(void){
int t[ELEMENTS],element;
int posicionsPlenes;
int index,esborrar;
bool trobat;
cout<<"Inserir elements: "<<endl;
inserirElements(t,index);
posicionsPlenes=index;
//inserin elements
cout<<"Esborrar un element? 0=no/1=si"<<endl;
cin>>esborrar;
while(esborrar==1 && posicionsPlenes >0){
  cout<<"Element a esborrar ? ";
  cin>>element;
  index=0;
  trobat=false;
  while(index < posicionsPlenes && !trobat){
     if(t[index] == element){
        trobat=true;
        int i = index;
        while( i < posicionsPlenes){          // esborrar element
           t[i]= t[i+1];
           i++;
        }
```

```
        posicionsPlenes--;
      }
    index++;
  }
  cout<<" El contingut de la taula es:"<<endl;
  mostrarElements(t);
  cout<<"Esborrar un altre element? 0=no/1=si"<<endl;
  cin>>esborrar;
}
if(posicionsPlenes == 0)cout<<"no queden elements a la taula";
cout<<endl;
return 0;
}
```

5.10 Taules de caràcters

Una taula de caràcters és una taula les components de la qual són de tipus *char*. En el món de la programació, hi ha molts casos en què és útil una taula de caràcters, per exemple, per guardar el nom o el cognom d'una persona o d'un conjunt de persones, el nom d'una ciutat, país, etc.

C++ proporciona tot un conjunt de funcions que permeten treballar amb taules de caràcters definides per l'usuari i que estan incloses en el fitxer capçalera *iostream*. Per aquest motiu, es dedica un apartat especial a aquest tipus de taules, per aprendre a dissenyar-les de manera que sigui possible aprofitar els avantatges o les prestacions que aquest llenguatge de programació ens proporciona.

H	o	l	a	\0					

Fig. 5.7 Taula de caràcters

En una taula de caràcters definida per l'usuari, si es volen utilitzar aquestes funcions, la informació introduïda ha d'anar seguida del caràcter null ('\0').

Vegeu com s'utilitzen aquestes funcions mitjançant un exemple:

Programa 69

Exemple per veure l'ús d'algunes funcions de C++ per a taules de caràcters

```
#include <iostream>
using namespace std;
int main()
{
char paraula[256];
cin>>paraula;
int longitud=strlen(paraula);
int i=0;
while(i<longitud){
    cout<<paraula[i];
    i++;
  }
cout<<endl;
cout<<"la longitud de la paraula és: "<<longitud<<endl;
if(strcmp(paraula,"Hola")==0) cout<<"la paraula introduida és "<<"Hola";
else cout<<"la paraula introduïda no és "<<"Hola"<<endl;;
```

```
strcpy(paraula,"Hola");
cout<<"ara a la taula de caràcters hi ha : "<<paraula<<endl;
i=0;  // tractament posició a posició
cout<<endl;
while(i<longitud){
    cout<<paraula[i];
    i++;
  }
i=0;
cout<<endl;
while(paraula[i]!='\0'){
    cout<<paraula[i];
    i++;
  }
cout<<endl;
return 0;
}
```

5.11 Ordenació dels elements d'una taula

Una operació que es fa molt sovint amb les taules, sobretot amb les d'una dimensió, és ordenar els seus elements. Hi ha molts mètodes per ordenar els elements d'una taula, per exemple: la bombolla, selecció, montecarlo, etc. En aquest capítol presentem els dos primers i a partir d'aquests es podrà entendre el funcionament de la resta de mètodes si és necessari.

5.11.1 Mètode de la bombolla

Consisteix a recórrer la taula de valors a ordenar i comparar-los de dos en dos. Si els elements estan ben ordenats, es passa a la següent parella, si no ho estan s'intercanvien i es passa al següent element fins a arribar al final de la taula. El procés complet es repeteix fins que els elements estiguin ordenats. Es veurà millor amb un exemple:

Suposem que es volen ordenar els següents elements en ordre creixent:

25, 3, 8, 6, 28, 1

Es comença comparant 25 i 3. Com que estan mal ordenats s'intercanvien i la llista quedarà:

3, 25, 8, 6, 28, 1

La següent parella és 25 i 8, s'intercanvien i segueix el procés...

En acabar las passada, la llista serà:

3, 8, 6, 25, 1, 28

Com que encara no estan ordenats del tot es fa una segona passada, però ara no és necessari recórrer tots els elements, ja que l'últim està ben ordenat i com que l'ordre és creixent, sempre serà el major, per tant no serà necessari incloure'l en la segona passada que serà:

3, 6, 8, 1, 25, 28

Ara és el 25 el que està a la posició correcta, la penúltima. Las successives pasades deixeran els elements:

3ª) 3, 6, 1, 8, 25, 28
4ª) 3, 1, 6, 8, 25, 28
5ª) 1, 3, 6, 8, 25, 28

El codi que hi ha a continuació implementa aquest mètode.

Programa 70

Mètode de la bombolla.

```
#include <iostream>
using namespace std;

int const TAMANY=100;

void mostrarVector(int H[],int posicionsPlenes){
 int i=0;
 while(i<posicionsPlenes){
    cout<<H[i];
    i++;
    cout<<endl;
 }
}

void omplirVector(int H[],int& i){
 i=0;
 cin>>H[i];
 while(H[i]!=-1.0 && i< TAMANY){
    i++;
    cin>>H[i];
 }
 if(i==TAMANY)H[i]=-1;
}
int main (){
int llista[TAMANY],i,j,temp,elements;
omplirVector(llista,elements);
for (i=1; i<elements; i++)
          for (j=0 ; j<elements - 1; j++)
             if (llista[j] > llista[j+1]){
                    temp = llista[j];
                    llista[j] = llista[j+1];
                    llista[j+1] = temp;
             }

mostrarVector(llista,elements);
system("pause");
return 0;
}
```

Si els elements d'entrada son:

$$4, 2, 5, 2, 1$$

Hi ha 5 elements, és a dir, TAMANY pren el valor de 5. Es compara el primer amb el segon element, 4 es més gran que 2, inrercanviem:

$$2, 4, 5, 2, 1$$

Ara es compara el segón amb el tercer, 4 és menor que 5, així que no s'ha de fer res. Continuem: 5 és més gran que 2. Intercanviem i tenim:

$$2, 4, 2, 5, 1$$

Comparem el quart i cinquè: 5 és més gran que 1. Intercaviem altre vegada:

$$2, 4, 2, 1, 5$$

Repetint el procés s'obté el següent resultat:

$$2, 2, 1, 4, 5$$
$$2, 1, 2, 4, 5$$

Però en aquest algorisme s'hi poden fer alguns canvis per millorar-ne el rediment.

Si s'observa el codi, es pot veure que en cada passada un dels elements queda en la posició definitiva. En la primera passada el 5 (element més gran) ha quedat en l'última posició, en la segona el 4 (el segon element més gran) en la penúltima posició, etc. Si en cada passada disminueix el nombre de comparacions, es pot evitar fer comparacions innecessàries? Si, tan sols cal canviar el *for* intern per aquest altre:

for (j=0; j<TAMANY - i; j++)

on i és la variable que indica el nombre de vegades que s'ha recorregut la taula comparant els elements de dos en dos. És possible que les dades quedin ordenades abans de completar el *for* extern. Si es vol optimitzar fins aquest punt, es pot modificar l'algorisme perque si en un recorregut no hi ha hagut cap canvi acabi l'execució, ja que significa que les dades ja estan ordenades. Una alternativa pot ser anar guardant l'última posició de la taula on s'ha fet un intercanvi, i en la següent passada només comparar fins la posició anterior a aquesta.

5.11.2 Mètode de selecció

Aquest mètode selecciona l'element més petit o més gran, depenent de l'ordre d'ordenació desitjat i l'intercanvia pel primer element. Seguidament fa el mateix amb els $n-1$ elements restants. Si un element està ja en la seva posició no es mou. Aquest mètode també ordena una taula de n elements en $n-1$ passades sobre la taula. Vegem-ne un exemple que permet ordenar com a molt 25 elements.

Programa 71

Mètode de selecció.

```
#include <iostream>
using namespace std;
```

```
#define N 25

int main(void){
    int v[N], i,j,imin,aux,num;
     cout<<"entra el nombre d'elements del vector";
     cin>>num;
     for( i=0;i<num;i++){
          cout<<"entra el seguent enter"<<endl;
          cin>>v[i];
     }
  for( i=0;i<num-1;i++){
     imin = i;
     for (j = i+1;j<num;j++){
           if  (v[j] < v[imin]){
                imin = j ;
           }
        }
        if (imin > i){
             aux = v[imin]; v[imin] = v[i]; v[i] = aux;
         }
   }
   for( i=0;i<num;i++){
           cout<<v[i]<<;
           }
   system("pause");
}
```

Programa 72

Mètode de selecció utilitzant accions i funcions en el disseny de l'algorisme.

```
#include <iostream>
using namespace std;

#define N 25
typedef int vector[N];
void LlegirVector(vector v, int &n);
void OrdenaVector (vector v, int n);
int Minim (vector v, int j, int n);
void EscriureVector (vector v,int n);
void Intercanvia(vector v, int i, int j);
int main(void){
   vector v; int  n;
   LlegirVector(v,n);
   OrdenaVector (v,n);
   EscriureVector (v,n);
  system("pause");
}
void LlegirVector(vector v, int &n){
     int i;
     cout<<"entra el nombre d'elements del vector";
     cin>>n;
```

```
     for( i=0;i<n;i++){
          cout<<"entra el seguent enter"<<endl;
          cin>>v[i];
     }
}
void OrdenaVector (vector v, int n){
  int i,imin;
  for( i=0;i<n-1;i++){
    imin = Minim(v,i,n);
    if(imin!= i) Intercanvia (v,i,imin);
}
}
int Minim (vector v, int i, int n){
  int j, imin;
     imin = i;
     for( j=i+1;j<n;j++){
           if  (v[j] < v[imin]){ imin = j;}
      }
    return  imin;
}
void Intercanvia(vector v, int i, int j){
int aux;
aux = v[j]; v[j]= v[i]; v[i] = aux;
}
void EscriureVector(vector v, int n){
     int i;
      for( i=0;i<n;i++){
           cout<<v[i]<<endl;
           }
}
```

5.12 Exemples dirigits

Completeu els programes següents:

1. Suposem que es vol conèixer l'evolució de l'índex de preus de consum (IPC) d'enguany encara que tot just estiguem a la meitat, per això es disposa d'una taula de dues dimensions anomenada $IPC[10][6]$, on les files representen els productes que són de consum habitual i les 6 columnes als 6 primers mesos de l'any als quals es fa referència, és a dir, des de gener fins a juny. A cada cel·la, $IPC[i][j]$, hi ha un nombre real que representa la contribució a aquest índex del producte i en el mes j. Es vol saber quina és la mitjana de la contribució de cada producte durant aquests primers 6 mesos de l'any i quin és el producte que ha aconseguit la màxima d'aquestes mitjanes.

   Per resoldre aquest problema, només cal que definiu una taula de dues dimensions pensant que, per exemple, les files representaran els mesos a estudiar i les columnes els productes. La resta són càlculs molt senzills.

2. Suposeu que s'ha encarregat a un fotògraf que faci un duplicat d'una fotografia antiga en blanc i negre que té molt de soroll (està molt deteriorada). Ell l'ha digitalitzada i n'ha obtingut una taula de dues dimensions T de 600 files per 800 columnes amb components reals, on l'element $T_{i,j}$ de la taula és el nivell de gris del píxel (punt) i, j de la fotografia, però el cas és que molts d'ells tenen un valor $-1{,}0$, que indica que no hi ha informació del nivell de gris, ja que aquest punt de la fotografia està deteriorat.

El fotògraf és un home que no es rendeix fàcilment; per això ha pensat que podria idear un programa que, per a cada píxel que no té informació del nivell de gris (que té un $-1{,}0$), el dedueixi calculant la mitjana del nivell de gris que tenen els seus veïns més propers (posicions contigües de la taula).

El programa és el que hi ha a continuació, però li falta la part del càlcul de la mitjana de cada posició de la taula, i per programar-la s'han de tenir en compte diferents casos:

- Els elements: $T_{0,0}$, $T_{0,columnes-1}$, $T_{files-1,0}$, $T_{files-1,columnes-1}$ només tenen dos veïns.
- La resta dels elements del perímetre tenen tres veïns. Per exemple: els de $T_{1,0}$ són $T_{0,0}$, $T_{1,1}$ i $T_{2,0}$,
- La resta en tenen quatre.

De manera que, per a cada píxel que no tingui informació del nivell de gris, li calculi el més adient depenent del que tenen els seus veïns.

```
#include <iostream>

using namespace std;

const int f=10;
const int c=10;

typedef int m[f][c];
int main()
{
  int i,j;
  m M;

//Aquest codi omple la matriu per provar el problema

  for (i=0;i<f;i++){
   for (j=0;j<c;j++){
    M[i][j]= -1;
   }
  }

  M[3][4]=12;
  M[5][4]=23;
  M[3][0]=3;
  M[0][4]=4;
  M[7][1]=12;
  M[3][9]=1;

  cout<< endl<<"matriu original"<<endl;
  for (i=0;i<f;i++){
      cout<< endl<<"fila"<<endl;
   for (j=0;j<c;j++){
    cout<<M[i][j]<<" ";
   }

  }

// Aquí hi ha d'anar el tractament del soroll
```

```
//Escriu la matriu per pantalla de  la fotografia restaurada

  cout<< endl<<"matriu sense soroll"<<endl;
  for (i=0;i<f;i++){
      cout<< endl<<" fila"<<endl;
   for (j=0;j<c;j++){
    cout<<M[i][j]<<" ";
   }
  }
  return 0;
}
```

3. Implementeu un programa que permeti calcular l'histograma del nombre d'alumnes que han aprovat una determinada assignatura els darrers 10 anys. Els diferents valors de les notes corresponents s'emmagatzemen en una matriu de grandària 10×90, on 90 representa el nombre màxim d'alumnes matriculats en un any, però pot passar que n'hi hagi menys. El programa ha de demanar a l'usuari que introdueixi les notes de cada any fins a un total de 10. Per acabar d'introduir les notes corresponents a un any, s'introduirà un -1, que també es guardarà a l'última posició de cada fila. En acabar, l'histograma s'ha d'imprimir per pantalla.

   - Per guardar les dades, s'ha de definir una taula de dues dimensions parcialment plena i controlada mitjançant sentinella. L'histograma es pot representar amb una taula d'una sola dimensió, on cada component guardi el nombre d'alumnes que han aprovat l'assignatura
   - Es proposa dividir el programa en tres accions:
     - Introduir les dades a la taula.
     - Calcular l'histograma, havent definit prèviament una taula d'una dimensió per guardar-hi les dades, és a dir, a cada posició hi ha d'haver el nombre d'aprovats per a una determinada nota.
     - Imprimir l'histograma per pantalla.

4. Dissenyeu un programa que permeti esbrinar si una paraula és un palíndrom, és a dir, que llegida d'esquerra a dreta és la mateixa que llegida de dreta a esquerra, per exemple *anilina*. Optimitzeu el codi tenint en compte que, si en un moment donat ja s'ha vist que no és un palíndrom, el programa no continuï comparant caràcters, és a dir, apliqueu un esquema de cerca. Una idea podria ser definir una funció ***bool palindrome(char p[256])*** que retornés ***true*** en cas que ho fos i ***false*** en cas contrari.

5. Amplieu el programa anterior perquè pugui tractar frases palíndroms com per exemple "*A Tirana mana Rita*" aprofitant la definició de la funció del problema anterior.

   En aquest cas, podeu llegir el text amb ***cin*** o ***cin.get*** depenent de si voleu llegir també el caràcter ' ' (esapi).

5.13 Exercicis proposats

1

Feu un programa que llegeixi deu valors enters des del teclat, els guardi en un vector, en calculi la suma, la mitjana dels valors, el màxim i el mínim, i escrigui aquests resultats per pantalla.

2

Feu un programa que calculi l'histograma del nombre d'alumnes que han aprovat una assignatura els darrers deu anys. Els diferents valors de les notes corresponents s'emmagatzemen en una matriu de grandària 10×90, on 90 representa el nombre màxim d'alumnes matriculats en un any, però pot passar que algun any no n'hi hagi tants de matriculats. El programa demanarà a l'usuari que introdueixi les notes corresponents a cada any fins a un total de 10, i per indicar que s'acabaran d'introduir les notes d'un any s'introduirà un -1, que també es guardarà a l'última posició de cada fila. El programa ha d'imprimir l'histograma per pantalla i l'any amb més i menys aprovats.

3

Feu un programa que, donada una paraula, esbrini si és o no un palíndrom. La decisió de si és palíndrom o no l'ha de prendre una funció amb el prototipus *bool Palindrome(char palabra[40]);*. La funció ha de tornar *true* si la paraula és un palíndrom i *false* en cas contrari. Una paraula és un palíndrom si, quan es llegeix des del final al principi, és igual que llegint-la del principi al final, per exemple: "CUC" o "ANNA". La funció no ha de fer distincions entre majúscules i minúscules. Feu el mateix programa utilitzant una acció.

4

Es vol conèixer l'evolució de l'índex de preus de consum (IPC), i per això es disposa d'una taula de dues dimensions anomenada *IPC[10][6]*, on les files representen els productes que són de consum habitual i les sis columnes als sis primers mesos de l'any als quals es fa referència, és a dir, des de gener fins a juny. A cada cel·la, *IPC[i][j]*, hi ha un real que representa la contribució a aquest índex del producte i el mes j. Es vol saber quina és la mitjana de contribució de cada producte durant els primers sis mesos de l'any i quin és el producte que ha aconseguit la màxima d'aquestes mitjanes. Dissenyeu una acció que calculi aquestes dades.

5

Escriviu un programa que guardi en un vector els mesos de l'any introduïts de forma ordenada (gener, febrer, març ...) per teclat.

6

Modifiqueu el programa anterior perquè els mesos de l'any es puguin introduir en qualsevol ordre.

7

Escriviu un programa que llegeixi una seqüència d'enters des del teclat i la mostri invertida per pantalla.

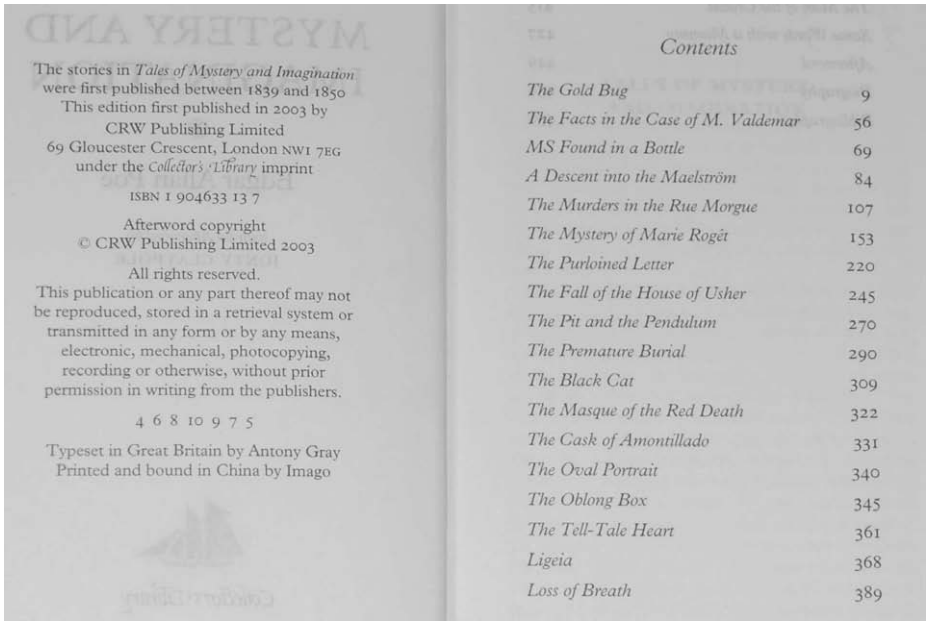
6 Estructures

6.1 Introducció

Com s'ha explicat al capítol 2, C++ ens obliga a especificar el tipus de dades de les variables en el moment de la declaració. Amb aquest fi, ens proporciona un conjunt de tipus de dades elementals (per exemple, *int*, *double*, *char*, *bool*, etc.), que fins ara han estat suficients per a la majoria de les nostres necessitats.

Malgrat tot, hi ha situacions en les quals és necessari disposar de *tipus de dades complexos* que ens permetin declarar variables complexes. Per exemple, considereu la figura 6.1. Si volem crear un programa que manipuli llibres, no ens és possible capturar tota la informació rellevant a un llibre en una única variable elemental (és a dir, *int*, *char*, *string*, etc.). Per exemple, suposem que ens interesa guardar, per a cada llibre, el títol, l'autor, l'ISBN i el nombre de pàgines que té. Amb el que hem vist fins ara, necessitaríem quatre variables diferents que, conceptualment, farien referència al mateix llibre.

> En representar objectes del món real en un programa informàtic, cal decidir quina informació ens és rellevant. Per exemple, l'editorial del llibre no ho seria per a l'exemple proposat.



The stories in *Tales of Mystery and Imagination* were first published between 1839 and 1850
This edition first published in 2003 by
CRW Publishing Limited
69 Gloucester Crescent, London NW1 7EG
under the *Collector's Library* imprint
ISBN 1 904633 13 7

4 6 8 10 9 7 5
Typeset in Great Britain by Antony Gray
Printed and bound in China by Imago

Contents

Fig. 6.1 Imatge d'un llibre. A l'esquerra es pot veure tota la informació relacionada amb el llibre. A la dreta, l'índex

L'ús de seqüències de valors o taules pot ser una solució per a determinats casos concrets, però en general són insuficients. Les seqüències (v. capítol 3) no són gaire flexibles i estan orientades a resoldre un tipus concret de problema (un flux de dades homogeni del qual no es coneix el final). Així doncs, en el cas de voler capturar la informació d'un llibre, no ens serien útils: no es tracta d'un flux de dades (en el sentit tradicional del terme) i, encara que ho fos, continuaríem sense poder agrupar la informació del llibre i, a més, atès que títol, autor i ISBN són dades de tipus *string* i el nombre de pàgines de tipus *enter*, disposaríem d'una heterogeneïtat de dades que no podem tractar amb seqüències. Per la seva banda, les taules (v. capítol 5) tampoc no permeten heterogeneïtat de dades i estan pensades per emmagatzemar, de forma conjunta, un seguit d'elements que podem referenciar de forma individual a partir d'un índex de referència.

Tot plegat, per a cobrir aquesta necessitat, C++ ens permet declarar estructures de dades pròpies (és a dir, nous tipus de dades) amb les quals podem agrupar diferent tipus d'informació sota un *embolcall semàntic*, tot i que aquesta informació sigui heterogènia.

És possible crear tipus de dades propis en C++.

6.2 Conceptes bàsics

El programador pot definir estructures de dades (també anomenades *tuples*) pròpies en C++. A l'hora de crear un nou tipus de dades, cal especificar els elements que les componen. Dit d'una altra forma, es declaren les variables que es volen agrupar conceptualment sota un mateix nom.

Declaració: La sintaxi que s'ha d'emprar per declarar una nova tupla (i, per tant, un nou tipus de dades) és la següent:

```
struct <nom>
{
   <tipus_dades_camp_1> <nom_camp_1>;
   <tipus_dades_camp_2> <nom_camp_2>;
   ...
   <tipus_dades_camp_N> <nom_camp_N>;
};
```

on *struct* és una paraula clau i *nom* és l'identificador de la nova estructura de dades. Entre claus, es declaren un conjunt de variables que conformaran els *camps* de la nova estructura. Observeu que es permet l'heterogeneïtat entre els membres de la tupla. A més, cal adonar-se que és obligatori l'ús del punt i coma darrere la clau de tancament, per a una compilació correcta. Formalment, una *tupla* o *estructura de dades* es defineix com un conjunt de variables (també conegudes com *camps* o *membres* de la tupla) agrupades sota un únic nom. Una vegada declarada una nova tupla, C++ crea un nou tipus de dades amb el mateix nom especificat a la declaració de la tupla i que es pot emprar al llarg del nostre programa.

Un nou tipus de dades es declara especificant un conjunt de variables que, conceptualment, ens interessa agrupar.

No oblideu el punt i coma (;) al final de la declaració d'un nou tipus de dades!

Per exemple, si volem definir un nou tipus de dades per guardar la informació d'un llibre, podem crear la tupla següent:

```
struct llibre {
    string titol;
    string autor;
    string ISBN;
    int num_pagines;
};
```

En aquest cas, hem declarat quatre camps: *titol*, *autor* i *ISBN* (de tipus *string*) i *num_pagines* (de tipus enter). Ara ja podem emprar aquest nou tipus de dades al nostre programa i, per tant, podem crear variables d'aquest tipus. Per exemple:

```
llibre meu_llibre;
```

que declara una variable de tipus *llibre* i de nom *meu_llibre*. Seguint amb la nomenclatura emprada al capítol 2, aquesta variable es pot representar a la memòria, tal com es mostra a la part esquerra de la figura 6.2. Allà, veiem que el nom de la variable (*meu_llibre*) conté tots els camps declarats al tipus de dades llibre.

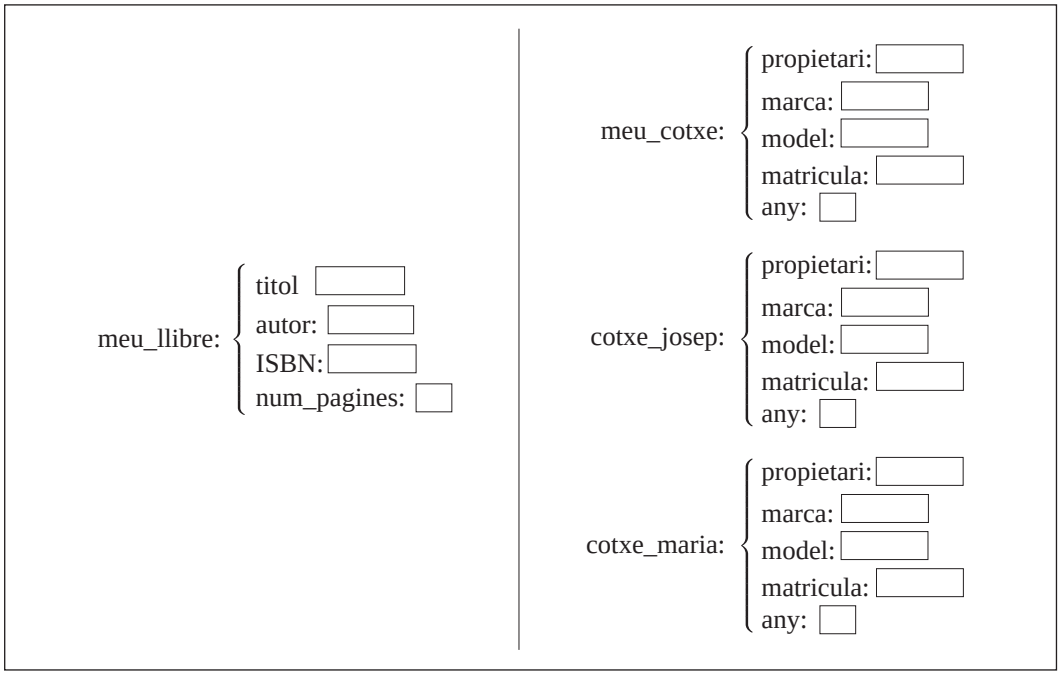


Fig. 6.2 Representació a memòria de diverses variables complexes: a l'esquerra, la variable *meu_llibre* de tipus llibre; a la dreta, les variables *meu_cotxe*, *cotxe_josep* i *cotxe_maria* de tipus cotxe.

Un altre exemple, en aquest cas per emmagatzemar la informació referent a un cotxe, seria:

```
struct cotxe {
    string propietari;
    string marca;
    string model;
    string matricula;
    int any;
}meu_cotxe, cotxe_josep, cotxe_maria;
```

La tupla *cotxe* conté cinc camps: *propietari*, *marca*, *model*, *matricula* i *any*).

Les variables que agrupem en el nostre nou tipus de dades poden ser heterogènies i fins i tot de tipus complexos prèviament definits (v. secció 6.5).

En definir una nova estructura, podem declarar, alhora, variables del nou tipus de dades especificant-ne els identificadors, just abans del punt i coma final (per exemple: *meu_cotxe*, *cotxe_josep* i *cotxe_maria*). Com es pot veure a la part dreta de la figura 6.2, aquestes tres variables contenen exactament els mateixos camps (heretats del tipus de dades cotxe), però cadascuna conté els seus propis (fixeu-vos que els camps es repliquen dins de cada variable). Per poder accedir als camps específics de cada variable, cal emprar l'operador punt '**.**', que es descriu a continuació.

Accés als camps de la tupla: L'operador punt '**.**' ens permet accedir a cada un dels camps definits dins de la tupla i operar amb ells com si fossin variables elementals dels seus tipus respectius. L'ús de l'operador punt és el següent:

```
<nom_variable>.<nom_camp>
```

Per exemple:

```
meu_cotxe.propietari
meu_cotxe.marca
meu_cotxe.model
meu_cotxe.matricula
meu_cotxe.any
```

Cada una d'aquestes expressions ens permet accedir als camps que conté la variable *meu_cotxe* (v. figura 6.2), i podem operar amb aquestes expressions com si fossin variables elementals. Així, *meu_cotxe.propietari*, *meu_cotxe.marca*, *meu_cotxe.model* i *meu_cotxe.matricula* són expressions de tipus *string*, mentre que *meu_cotxe.any* és de tipus *enter*. Per tant, aquestes expressions es poden emprar com si fossin variables d'aquests tipus.

Tot i que podem declarar tipus de dades complexes, C++ només treballa amb tipus elementals (per exemple: *int*, *double*, *char*, *bool*, *string*, etc.) i, per tant, a l'hora de manipular tipus de dades complexos, cal fer-ho a nivell elemental accedint als seus camps emprant l'operador '.'

Recordeu: les expressions obtingudes amb l'operador '.' es poden emprar com si fossin variables elementals del tipus en qüestió.

Programa 73

Programa que demana informació a una tercera persona sobre el seu cotxe i li mostra per pantalla les seves dades i les del nostre cotxe.

En aquest exercici, emprem l'estructura cotxe anterior. A més, necessitem dues variables de tipus cotxe (*meu_cotxe* i *teu_cotxe*) que ens permetin emmagatzemar tota la informació del nostre cotxe i del cotxe de la persona emprant el programa. En aquest exemple, demanem per teclat tota la informació del cotxe de l'usuari i l'anem guardant a la variable *teu_cotxe*. A més, cal inicialitzar la variable *meu_cotxe* amb les dades del nostre cotxe. Finalment, mostrarem per pantalla la informació continguda en totes dues variables.

```
#include <iostream>
using namespace std;
struct cotxe {
    string propietari;
    string marca;
    string model;
    string matricula;
    int any;
};
int main(void) {
  cotxe meu_cotxe, teu_cotxe;
  //Emplenem amb valors els camps de la variable meu_cotxe
  meu_cotxe.propietari = "Kimi";
  meu_cotxe.marca = "Ferrari";
  meu_cotxe.model = "F430 Spider";
  meu_cotxe.matricula = "0001-FER";
  meu_cotxe.any = 2006;
  /* Emplenem amb valors els camps de la variable
  teu_cotxe (demanant-los per teclat) */
  cout << "Quin nom tens? ";
  cin >> teu_cotxe.propietari;
  cout << "Quina marca de cotxe tens? ";
  cin >> teu_cotxe.marca;
  cout << "Quin model es? ";
  cin >> teu_cotxe.model;
  cout << "Quina matricula te? ";
  cin >> teu_cotxe.matricula;
  cout << "De quin any es?: ";
  cin >> teu_cotxe.any;
  //Treiem totes les dades per pantalla!
  cout << "Be, " << teu_cotxe.propietari;
  cout <<", el teu cotxe es un " << teu_cotxe.marca << " ";
  cout << teu_cotxe.model << " de l'any ";
  cout << teu_cotxe.any << ". La seva matricula es: ";
  cout << teu_cotxe.matricula << ". Quin cotxe mes maco!";
  cout << endl << endl;
  cout << "Jo em dic " << meu_cotxe.propietari;
  cout <<", i tinc un " << meu_cotxe.marca;
  cout << " " << meu_cotxe.model << " de l'any ";
  cout << meu_cotxe.any << ". La seva matricula es: ";
  cout << meu_cotxe.matricula << "." << endl;
  system("pause");
}
```

> Com a única excepció, dues variables no elementals (és a dir, de tipus complexos) poden comparar-se o assignar-se directament, sempre que no continguin ni taules ni matrius.

En general, C++ només treballa amb tipus de dades elementals (v. capítol 2). En el cas de treballar amb estructures (on diverses variables s'agrupen sota un mateix embolcall semàntic), necessitem accedir als seus elements (mitjançant l'operador punt) i manipular-los per separat.

L'operador *Typedef*: C++ també ens permet definir els nostres propis tipus de dades com a àlies de tipus de dades ja existents gràcies a l'operador *typedef*. La sintaxi que s'ha d'emprar és la següent:

```
typedef <tipus_existent> <nou_nom>;
```

on *tipus_existent* és un tipus elemental o complex (és a dir, declarat amb la paraula clau *struct*) i *nou_nom* és l'identificador del nou tipus. Per exemple:

```
typedef double nota;
typedef char lletra;
typedef int taula[MAX_ELEMS];
```

En aquest exemple, hem definit tres nous tipus de dades; *nota* com un *double*, *lletra* com un *char* i *taula* com una taula d'enters de longitud màxima `MAX_ELEMS`. Ara, podem declarar variables d'aquests tipus:

```
nota nota_josep, nota_andres;
lletra primera_lletra, ultima_lletra;
taula meva_taula;
```

Fixeu-vos que *typedef* no declara un nou tipus de dades, sinó que crea un *àlies* per a un tipus ja existent. *Nota_josep* pot considerar-se de tipus *nota* o *double*, perquè de fet són el mateix tipus i, per tant, es poden emprar com a sinònims. *Typedef* és realment útil per adaptar els tipus de dades preexistents de C++ a les necessitats semàntiques del programador; això permet donar-los noms més propers al que representaran.

> L'operador *typedef* ens permet crear àlies (és a dir, *tipus de dades sinònims*) de tipus de dades ja existents.

Combinar *typedef* i *struct*: Finalment, és bastant freqüent trobar *typedef* en conjunció amb la paraula clau *struct*. De fet, és la forma més extesa per definir noves estructures en C++, tot i que és redundant:

```
 typedef struct{
    string propietari;
    string marca;
    string model;
    string matricula;
    int any;
}cotxe;
```

Observeu que aquesta sintaxi és fruit de la composició de la sintaxi d'ambdues paraules clau; ja que es tracta d'un *typedef*, on el *tipus_existent* és una estructura declarada al moment (emprant l'*struct*) i *nou_nom* és l'identificador elegit per a la nova estructura (és a dir, *cotxe*). A tots els efectes, aquesta definició és equivalent a la que s'ha presentat a la resolució de l'enunciat 73.

> Tot i ser redundant, la forma més estesa de declarar un nou tipus de dades és combinant les paraules clau *typedef* i *struct*.

6.3 Exemples dirigits

Programa 74 *Cal fer un programa que llegeixi dues coordenades cartesianes i digui si són iguals.*

Cal emplenar els tres forats de manera que, un cop demanades les coordenades dels dos punts per teclat, el programa respongui que són iguals si són exactament el mateix punt, i que són diferents altrament.

```
struct punt {
 double x, y;
};
int main(void) {
 punt a, b;
 cout << "Introduiu les coordenades del primer punt: ";
 cin >> a.x >> [          ];
 cout << "Introduiu les coordenades del segon punt: ";
 cin >> [          ];
 if([          ])
    cout << "Ambdos punts son iguals." << endl;
 else
    cout << "Els punts son diferents." << endl;
 system("pause");
}
```

Programa 75 *Cal fer un programa que sumi una seqüència de nombres complexos acabada en* $0+0i$.

Aquest és un enunciat clàssic de seqüències i, per tant, cal fer ús de l'esquema de recorregut vist al capítol 2. En general, podem combinar les tuples amb totes les eines vistes al llarg d'aquest llibre; per exemple, en aquest cas, amb seqüències.

Empleneu els set forats de manera que sumi un nombre indefinit de nombres complexos (és a dir, una seqüència de nombres que té per sentinella el $0+0i$) i en calculi la suma. Cal declarar el tipus *complex* i emprar-lo per resoldre l'exercici.

```
#include <iostream>
using namespace std;
typedef struct {
     [          ] re, im;
} complex;
int main (void) {
  complex c, resultat;
  resultat.re = [          ]; resultat.im = [          ];
  cout << "Introduiu un nombre complex (part real i " <<
  "part imaginaria). Per acabar, introduiu el 0.0 0.0 : " << endl;
  cin >> c.re; cin >>c.im;
  while (c.re [          ] 0.0 && c.im [          ] 0.0) {
     resultat.re = resultat.re + c.re;
     resultat.im = resultat.im + c.im;
  cout << "Introduiu el seguent complex (part real i " <<
  "part imaginaria). Per acabar, introduiu el 0.0 0.0 : " << endl;
     cin >> [          ]; cin >> [          ];
  }
  cout << "La suma dels nombres introduits es " << resultat.re << "+"
  << resultat.im << "i" << endl;
  system("pause");
}
```

Programa 76 *Cal crear un programa per emmagatzemar el resultat del campionat espanyol de fútbol de primera divisió a la temporada 2008-2009.*

Es demana emplenar els nou forats del codi de manera que el programa proposat defineixi correctament una estructura per emmagatzemar el resultat d'una lliga de fútbol i s'empleni amb les dades de la primera divisió espanyola a la temporada 2008-2009.

```
#include <iostream>
using namespace std;
const NUM_EQUIPS = 20;
struct equip{
    [          ] nom;
    int punts;
    [          ] golsFavor;
    [          ] golsContra;
};
int main (void) {
  [          ] lliga[NUM_EQUIPS];
  for(int i = [          ]; i < NUM_EQUIPS; i++)
  {
      cout << "Introduiu el nom de l'equip que va acabar "
      "a la posicio " << i+1 << endl;
      cin >> lliga[i].nom;
      cout << "Quants punts te?" << endl;
      cin >> [          ]
      cout << "I quants gols va marcar?" << endl;
      cin >> [          ]
      cout << "Per ultim, quants gols li van marcar?" << endl;
      cin >> [          ]
  }
  cout << "El campio va ser " << [          ]
  cout << ". Felicitats!! " << endl;
  system("pause");
}
```

6.4 Errors freqüents

En aquesta secció, fem èmfasi en els errors més freqüents a l'hora de treballar amb tuples. Intenteu resoldre els exercicis abans de llegir la solució:

Error 5.1. Sou capaços de trobar l'error en la declaració de la tupla *cotxe*?

```
struct cotxe {
    string propietari;
    string marca;
    string model;
    string matricula;
    int any;
}
```

Solució. Si intenteu compilar aquest tros de codi, rebreu aquest tipus d'error: *"new types may not be defined in a return type"*. De fet, és un error subtil, però molt comú a l'hora de declarar noves estructures. Recordeu que és obligatori posar ; en tancar l'última clau de l'*struct*.

Error 5.2. Perquè no compila aquest tros de codi?

```
#include <iostream>
using namespace std;
struct cotxe {
    string propietari;
    string marca;
    string model;
    string matricula;
    int any;
};
int main(void) {
  cotxe un_cotxe;
  cotxe.propietari = "Antoni";
  cotxe.marca = "Audi";
  cotxe.model = "A6";
  cotxe.matricula = "B-8976-AX";
  cotxe.any = 2009;
  system("pause");
}
```

Solució. L'error que reporta el compilador ens dirà: *expected primary-expression before '.' token*. Aquest error (molt comú) apareix en emprar el nom del tipus de dades a l'esquerra de l'operador punt en lloc del nom de la variable. Fixeu-vos que hauria de ser *un_cotxe.propietari = "Antoni"* i no *cotxe.propietari = "Antoni"*. Recordeu que cal declarar variables dels nous tipus de dades i accedir als camps de les variables.

Error 5.3. I aquesta declaració del *typedef*, quin problema té?

```
#include <iostream>
using namespace std;
int main(void) {
  sou meu_sou;
  cout << "Quin sou us agradaria tenir?" << endl;
  cin >> meu_sou;
  if(meu_sou > 30000)
    cout << "Carai! a mi tambe!" << endl;
  else
    cout << "No esta malament..." << endl;
  system("pause");
}
typedef sou double;
```

Solució. L'error que reporta el compilador és el següent: *sou undeclared (first use this function)*. Això passa perquè, tant si emprem *struct* com *typedef*, les declaracions de nou tipus han d'aparèixer abans d'emprar-les. És a dir, acostumeu-vos a declarar-les just sota els *includes* i la definició del *namespace* per evitar problemes.

Error 5.4. I aquest altre codi? Sou capaços de trobar els dos errors que hi ha?

```
#include <iostream>
using namespace std;
struct cotxe {
```

```
    string propietari;
    string marca;
    string model;
    string matricula;
    int any;
};
int main(void) {
  cotxe un_cotxe;
  un_cotxe = "Antoni";
  un_cotxe.marca = 5;
  //Resta d'instruccions (...)
  system("pause");
}
```

Solució. El primer error fa que el compilador ens retorni el missatge següent: *no match for 'operator =' in '"un_cotxe = Antoni"'*. Mentre que l'altre error no l'identifica el compilador. Els dos errors són fruit d'un mal ús de l'operador '.' dins del nostre codi. *un_cotxe* és una variable de tipus *cotxe* i, per tant, complexa. Per aquest motiu, no li podem fer una assignació d'una constant de tipus string *"Antoni"* (això dóna el missatge d'error comentat prèviament). Hem d'emprar l'operador punt i arribar a un camp de tipus *string* per tal de poder fer aquesta assignació (per exemple, *un_cotxe.propietari = "Antoni";*). El segon error és més subtil i no el detecta el compilador. *Un_cotxe.marca* és una expressió de tipus *string* i, per tant, requereix fer-li assignacions amb constants de tipus *string* (no enteres com a l'exemple). Com a norma, cal sempre tenir en compte quin tipus de dades avalua una expressió que empri l'operador punt per tal de tractar-la consegüentment.

6.5 Problemes avançats

En aquesta secció, presentem usos avançats de les tuples. Concretament, fem èmfasi en tuples que contenen camps que al seu torn també són tipus complexos, la combinació de tuples amb vectors, i com s'han d'emprar a l'hora de passar-les com a paràmetres de funcions o accions.

Estructures niades: Com hem explicat prèviament, en la definició d'un nou tipus de dades hem de declarar aquelles variables que volem agrupar dins de la nostra estructura. Per tant, és factible declarar tipus de dades que contenen variables complexes (sempre que s'hagi declarat abans), com es mostra a continuació:

```
typedef struct {
   int  hores, minuts, segons;
}temps;

typedef struct {
   int  any, mes, dia;
   temps hora;
} data;
```

En aquest exemple, el tipus de dades *data* està compost per tres camps enters (*any*, *mes* i *dia*) i un camp *hora* de tipus *temps*. *Temps* és un tipus de dades que l'usuari ha definit prèviament i que està compost, al seu torn, de tres camps enters (*hores*, *minuts* i *segons*).

En general, les estructures poden niar-se tant com sigui necessari, però a l'hora de manipular-les hem de tenir en compte que C++ ens exigeix treballar amb dades elementals, tal com hem vist prèviament en aquest capítol. Per tant, si volem declarar una variable de tipus *data* i inicialitzar-la, hi hem de treballar de la forma següent:

```
int main(void) {
  data data_actual;
  data_actual.any = 2009;
  data_actual.mes = 10;
  data_actual.dia = 26;
  data_actual.hora.hores = 19;
  data_actual.hora.minuts = 16;
  data_actual.hora.segons = 20;
}
```

És a dir, donat que *hora* és un tipus de dades complex, no hi podem treballar directament i necessitem accedir als seus camps, que sí són de tipus de dades elementals (en aquest cas, *ints*).

> Si una estructura conté un camp que, al seu torn, és d'un tipus de dades complex, cal continuar aplicant l'operador '.' fins a obtenir expressions que avaluïn tipus de dades elementals.

Si una funció retorna un tipus de dades complex, cal tenir en compte que l'operador = només es pot emprar per a estructures que no contenen ni taules ni matrius. Si no, serà necessari declarar una acció de còpia.

Pas per paràmetres: Les funcions i accions poden passar estructures com a paràmetres, tant per valor com per referència, seguint les mateixes regles que per a la resta de tipus elementals: les estructures passades per valor no conservaran els canvis fets dins de la funció / acció mentre que les passades per referència sí (v. capítol 3). També es poden declarar funcions que tinguin per tipus de retorn una estructura definida per l'usuari. El programa següent exemplifica les possibilitats del pas per paràmetres i retorn de valors d'una funció / acció.

> Atesa la mida potencialment gran de les estructures, quan cal passar una tupla com a paràmetre per valor és recomanable emprar el pas referència i l'operador *const* (v. capítol 4) en lloc de fer el pas per valor.

Programa 77 *Cal fer un programa que, donades les dades de dues persones, sigui capaç de distingir quina de les dues és més gran i quina pesa més.*

Per resoldre aquest exercici, necessitem crear el tipus de dades persona que emmagatzemarà les dades necessàries. Un cop fet, crearem dues accions (una per llegir les dades d'una persona per teclat i guardar-ho a una variable de tipus persona, i l'altra per mostrar per pantalla la informació continguda en una variable persona). Finalment, crearem dues funcions, que retornaran tota la informació de la persona més gran i de la que pesa més, respectivament.

```
#include <iostream>
using namespace std;
typedef struct {
  string nom;
  string cognom1;
  string cognom2;
  int edat;
```

```
  float pes;
  string professio;
}persona;
void llegeix_persona(persona &p);
void escriu_persona(const persona &p);
persona major_edat(const persona &pers, const persona &pers1);
persona major_pes(const persona &pers, const persona &pers1);
int main(void) {
  persona p, p1;
  int opcio;
  bool fi = false;
  cout << "Aquest senzill programa llegeix les dades de dues ";
  cout << "persones i es capac de saber quina de les dues es ";
  cout << "mes gran i quina pesa mes." << endl << endl;
  llegeix_persona(p);
  cout << endl;
  llegeix_persona(p1);
  cout << endl;
  cout << "Les dades de les persones que tinc son: " << endl;
  escriu_persona(p);
  cout << " i ";
  escriu_persona(p1);
  cout << endl;
  while(!fi)
  {
    cout << endl;
    cout << "A veure, que voleu saber? " << endl;
    cout << "1. Qui te mes edat." << endl;
    cout << "2. Qui pesa mes." << endl;
    cout << "3. Sortir." << endl;
    cout << "Introdueix el nombre que precedeix a l'opcio desitjada: ";
    cin >> opcio;
    while(opcio < 1 || opcio > 3)
    {
      cout << "Opcio incorrecta, si us plau, introduiu el nombre ";
      cout << "que precedeix l'opció desitjada: ";
      cin >> opcio;
    }
    switch(opcio)
    {
      case 1:
        cout << "La persona de mes edat es: " << endl;
        escriu_persona(major_edat(p,p1));
      break;
      case 2:
        cout << "La persona de mes pes es: " << endl;
        escriu_persona(major_pes(p,p1));
      break;
      default:
        fi = true;
    }
  }
  cout << "Adeu!" << endl;
```

```
  system("pause");
}
void llegeix_persona(persona &p) {
  cout << "Quin es el nom de la persona? ";
  cin >> p.nom;
  cout << "Quin es el seu primer cognom? ";
  cin >> p.cognom1;
  cout << "Quin es el segon cognom? ";
  cin >> p.cognom2;
  cout << "Quina edat te? ";
  cin >> p.edat;
  cout << "Quant pesa? ";
  cin >> p.pes;
  cout << "De que treballa? ";
  cin >> p.professio;
}
void escriu_persona(const persona &p) {
  cout << p.nom << " " << p.cognom1 << " ";
  cout << p.cognom2 << endl;
  cout << "Treballa com a " << p.professio << endl;
  cout << "Te " << p.edat << "anys i pesa ";
  cout <<  p.pes << " kilograms." << endl;
}
persona major_edat(const persona &pers, const persona &pers1) {
    if(pers.edat > pers1.edat)
        return pers;
    else
        return pers1;
}
persona major_pes(const persona &pers, const persona &pers1) {
    if(pers.pes > pers1.pes)
        return pers;
    else
        return pers1;
}
```

Taules d'estructures: Un cas pràctic bastant comú, per la seva versatilitat i potència, és la combinació de taules amb estructures. Una vegada definit un nou tipus de dades, aquest es pot emprar per especificar el tipus dels components d'una taula. Per exemple, podem definir una variable *cotxes_tenda* de tipus *taula*, que contingui 100 elements de tipus *cotxe* (és a dir, una taula de tuples) per a referir-nos a una col·lecció de cotxes (en aquest cas, de 100 cotxes com a màxim). Per exemple, suposem un concessionari de venda de cotxes de segona mà interessat a emmagatzemar en un sistema informàtic els cotxes de què disposa.

Programa 78 *Cal fer un programa que permeti emmagatzemar tots els cotxes que un concessionari de segona mà té.*

Per resoldre aquest exercici, definirem el tipus cotxe amb totes les dades necessàries. Un cop fet això, crearem un vector de 100 cotxes (nombre màxim de cotxes al concessionari) i anirem emplenant-lo amb informació rebuda per teclat.

```
#include <iostream>
using namespace std;
#define TAMANY 100
```

```
typedef struct {
    string nom;
    string cognom1;
    string cognom2;
    string nif;
    string telefon;     //Per a permetre nombres alfanumèrics
} persona;
typedef struct {
    persona propietari; //Qui els va vendre el cotxe
    string marca;
    string model;
    string matricula;
    int any;            //Any de matriculació
} cotxe;
void afegir_cotxe(cotxe c[], int &pos_lliure);
void cerca_cotxe(cotxe c[]);
void escriu_cotxe(const cotxe &c);
int main(void) {
  int opcio, i;
  int p_lliure = 0; //La primera posició lliure a la taula
  cotxe stock_cotxes[TAMANY];
  bool fi = false;
  while(!fi)
  {
    cout << "Programa de gestio del concessionari." << endl;
    cout << "1. Donar d'alta un cotxe." << endl;
    cout << "2. Sortir." << endl;
    cin >> opcio;
    while(opcio < 1 || opcio > 2)
    {
      cout << "Opcio incorrecta, torneu-ho a provar: ";
      cin >> opcio;
    }
    switch(opcio)
    {
      case 2:
        fi = true;
      break;
      default:
        if(p_lliure >= TAMANY)
        {
          cout << "Error. La taula esta plena.";
          cout << "No poden donar-se d'alta nous cotxes!";
          cout << endl;
        }
        else
          afegir_cotxe(stock_cotxes, p_lliure);
    }
  }
  cout << endl;
  if(p_lliure)
  {
    cout << "Aquests son els cotxes que s'han donat d'alta:";
```

```
    cout << endl;
    for(i=0; i<p_lliure; i++)
      escriu_cotxe(stock_cotxes[i]);
  }
  else {
    cout << "No hi havia cap cotxe al sistema.";
    cout << endl;
  }
  system("pause");
}
void afegir_cotxe(cotxe c[], int &pos_lliure) {
  cout << endl;
  cout << "Dades de l'expropietari: ";
  cout << endl << endl;
  cout << "Nom de l'antic propietari: ";
  cin >> c[pos_lliure].propietari.nom;
  cout << "Primer cognom: ";
  cin >> c[pos_lliure].propietari.cognom1;
  cout << "Segon cognom: ";
  cin >> c[pos_lliure].propietari.cognom2;
  cout << "NIF: ";
  cin >> c[pos_lliure].propietari.nif;
  cout << "Telefon: ";
  cin >> c[pos_lliure].propietari.telefon;
  cout << endl;
  cout << "Dades del cotxe: " << endl << endl;
  cout << "Marca del cotxe: ";
  cin >> c[pos_lliure].marca;
  cout << "Model: ";
  cin >> c[pos_lliure].model;
  cout << "Matricula: ";
  cin >> c[pos_lliure].matricula;
  cout << "Any de matriculacio: ";
  cin >> c[pos_lliure].any;
  cout << endl;
  /* Abans d'acabar incrementem pos_lliure, perquè
  aquesta posició del vector ja està plena. */
  pos_lliure++;
}
void escriu_cotxe(const cotxe &c) {
  cout << c.marca << " " << c.model;
  cout << " amb matricula: " << c.matricula << endl;
  cout << "Venut per " << c.propietari.nom;
  cout << " " << c.propietari.cognom1;
  cout << " " << c.propietari.cognom2;
  cout << " amb NIF: " << c.propietari.nif;
  cout << " y telefon: " << c.propietari.telefon;
  cout << endl;
}
```

Les tuples es poden combinar amb tots els conceptes i totes les eines vistes al llarg d'aquest llibre (tals com seqüències o taules).

Programa 79

Implementeu un programa que llegeixi per pantalla estructures de tipus estudiant i, tot seguit, les escrigui per pantalla:

```
/* Zona d'inclusions */
#include <iostream>
using namespace std;
typedef struct {
  string cognom1, cognom2, nom;
}nom_i_cognoms;
typedef struct {
  int any, mes, dia;
}data;
typedef struct {
  nom_i_cognoms nom_complet;
  data data_naixement;
  int nombre_matricula;
}estudiant;
/* Zona de prototipus */
void llegeix_estudiant(estudiant &e);
void escriu_estudiant(const estudiant &e);
/* Zona de definició de la funció principal (main) */
int main (void) {
  estudiant e1;
  llegeix_estudiant(e1);
  escriu_estudiant(e1);
  system("Pause");
}
/* Zona de definició de funcions auxiliars */
/*   Acció: llegeix_estudiant */
void llegeix_estudiant(estudiant& e) {
  cout << "Introduiu el nom de l'estudiant (e.g., Jose): ";
  cin >> e.nom_complet.nom;
     cout << endl << "Introduiu el primer cognom: ";
     cin >> e.nom_complet.cognom1;
     cout << endl << "Introduiu el segon cognom: ";
     cin >> e.nom_complet.cognom2;
     cout << endl << "Introduiu la data de naixement (any, mes, dia): ";
     cin >> e.data_naixement.any;
     cin >> e.data_naixement.mes;
     cin >> e.data_naixement.dia;
     cout << endl << "Introduiu el nombre de matricula: ";
     cin >> e.nombre_matricula;
   cout << endl << endl;
}
/*   Acció: escriu_estudiant   */
void escriu_estudiant(const estudiant &e) {
  cout << endl <<
          e.nom_complet.nom << " " <<
          e.nom_complet.cognom1 << " " <<
```

```
        e.nom_complet.cognom2 << endl;
  cout << e.data_naixement.dia << " <<
        e.data_naixement.mes << " <<
        e.data_naixement.any << endl;
  cout << "Nombre de matricula: " <<
        e.nombre_matricula << endl << endl;
}
```

Programa 80 *Es demana dissenyar un programa que llegeixi una seqüència d'estudiants per teclat i els emmagatzemi en una taula:*

```
/* Zona d'inclusions */
#include <iostream>
using namespace std;
/* Zona de definició de constants */
#define LongTaula 100
typedef struct {
  string cognom1, cognom2, nom;
}nom_i_cognoms;
typedef struct {
  int any, mes, dia;
}data;
typedef struct {
  nom_i_cognoms nom_complet;
  data data_naixement;
  int nombre_matricula;
}estudiant;
/* Zona de prototipus */
void llegeix_estudiant(estudiant& e);
void escriu_estudiant(const estudiant &e);
void copia_estudiant(estudiant& copia, const estudiant &e);
void sequencia_a_vector(estudiant t[LongTaula], int& n);
void escriu_vector(estudiant t[LongTaula], int n);
/* Zona de definició de la funció principal (main) */
int main (void) {
  estudiant t[LongTaula];
  int n;
  sequencia_a_vector(t,n);
  escriu_vector(t,n);
  system("pause");
}
/* Zona de definició de funcions auxiliars */
/*   Acció: llegeix_estudiant */
void llegeix_estudiant(estudiant& e) {
  cout << "Introdueix el nom de l'estudiant (e.g., Jose): ";
  cin >> e.nom_complet.nom;
  if (e.nom_complet.nom!="fi") {
     cout << endl << "Introdueix el primer cognom: ";
     cin >> e.nom_complet.cognom1;
     cout << endl << "Introdueix el segon cognom: ";
     cin >> e.nom_complet.cognom2;
     cout << endl << "Introdueix la data de naixement (any, mes i dia): ";
     cin >> e.data_naixement.any;
```

```
    cin >> e.data_naixement.mes;
    cin >> e.data_naixement.dia;
    cout << endl << "Introdueix el nombre de matricula: ";
    cin >> e.nombre_matricula;
  }
  cout << endl << endl;
}
/*   Acció: escriu_estudiant    */
void escriu_estudiant(const estudiant &e) {
  if (e.nom_complet.nom!="fi") {
  cout << endl <<
          e.nom_complet.nom << " " <<
          e.nom_complet.cognom1 << " " <<
          e.nom_complet.cognom2 << endl;
  cout << e.data_naixement.dia << " " <<
          e.data_naixement.mes << " " <<
          e.data_naixement.any << endl;
  cout << "nombre de matricula: " <<
          e.nombre_matricula << endl << endl;
  }
}
/*   Acció: còpia_estudiant    */
void copia_estudiant(estudiant& copia, const estudiant &e) {
    copia.nom_complet.nom=e.nom_complet.nom;
    copia.nom_complet.cognom1=e.nom_complet.cognom1;
    copia.nom_complet.cognom2=e.nom_complet.cognom2;
    copia.data_naixement=e.data_naixement;
    copia.nombre_matricula=e.nombre_matricula;
}
/*   Accio: seqüència_a_vector */
void sequencia_a_vector(estudiant t[LongTaula], int& n) {
   estudiant e;
   llegeix_estudiant(e);
   n=0;
   while (n<LongTaula && e.nom_complet.nom!="fi") {
     copia_estudiant(t[n],e); n++;
     llegeix_estudiant(e);
   }
}
/* Acció: escriu_vector */
void escriu_vector(estudiant t[LongTaula], int n) {
  int i;
  for (i=0; i<n; i++) escriu_estudiant(t[i]);
}
```

Programa 81 *Cal implementar un programa que llegeixi una seqüència d'estructures de tipus estudiant, n'emmagatzemi els elements en un vector i els ordeni creixentment per nombre de matrícula emprant el mètode d'ordenació per selecció:*

```
/* Zona d'inclusions */
#include <iostream>
using namespace std;
```

```
/* Zona de definició de constants */
#define LongTaula 100
typedef struct {
  string cognom1, cognom2, nom;
}nom_i_cognoms;
typedef struct {
  int any, mes, dia;
}data;
typedef struct {
  nom_i_cognoms nom_complet;
  data data_naixement;
  int nombre_matricula;
}estudiant;
/* Zona de prototipus */
void llegeix_estudiant(estudiant& e);
void escriu_estudiant(const estudiant &e);
void copia_estudiant(estudiant& copia, const estudiant &e);
void sequencia_a_vector(estudiant t[LongTaula], int& n);
void escriu_vector(estudiant t[LongTaula], int n);
void intercanvia_estudiant(estudiant& e1, estudiant& e2);
void ordena_mat(estudiant t[LongTaula], int n);
bool menor_mat(const estudiant &e1, const estudiant &e2);
/* Zona de definició de la funció principal (main) */
int main (void) {
  estudiant t[LongTaula];
  int n;
  sequencia_a_vector(t,n);
  ordena_mat(t,n);
  escriu_vector(t,n);
  system("Pause");
}
/* Zona de definició de funcions auxiliars */
/*   Acció: llegeix_estudiant */
void llegeix_estudiant(estudiant& e) {
  cout << "Introduiu el nom de l'estudiant (e.g., Jose): ";
  cin >> e.nom_complet.nom;
  if (e.nom_complet.nom != "fi") {
     cout << endl << "Introduiu el primer cognom: ";
     cin >> e.nom_complet.cognom1;
     cout << endl << "Introduiu el segon cognom: ";
     cin >> e.nom_complet.cognom2;
     cout << endl << "Introduiu la data de naixement (any, mes i dia): ";
     cin >> e.data_naixement.any;
     cin >> e.data_naixement.mes;
     cin >> e.data_naixement.dia;
     cout << endl << "Introduiu el nombre de matricula: ";
     cin >> e.nombre_matricula;
   }
   cout << endl << endl;
}
/*   Acció: escriu_estudiant   */
void escriu_estudiant(const estudiant &e) {
  if (e.nom_complet.nom != "fi") {
```

```
  cout << endl <<
          e.nom_complet.nom << " " <<
          e.nom_complet.cognom1 << " " <<
          e.nom_complet.cognom2 << endl;
  cout << e.data_naixement.dia << " " <<
          e.data_naixement.mes << " " <<
          e.data_naixement.any << endl;
  cout << "nombre de matricula: " <<
          e.nombre_matricula << endl << endl;
  }
}
/*   Acció: còpia_estudiant    */
void copia_estudiant(estudiant& copia, const estudiant &e) {
    copia.nom_complet.nom=e.nom_complet.nom;
    copia.nom_complet.cognom1=e.nom_complet.cognom1;
    copia.nom_complet.cognom2=e.nom_complet.cognom2;
    copia.data_naixement=e.data_naixement;
    copia.nombre_matricula=e.nombre_matricula;
}
/*   Acció: seqüència_a_vector    */
void sequencia_a_vector(estudiant t[LongTaula], int& n) {
   estudiant e;
   llegeix_estudiant(e);
   n=0;
   while (n<LongTaula && e.nom_complet.nom != "fi") {
     copia_estudiant(t[n],e); n++;
     llegeix_estudiant(e);
   }
}
/*   Acció: escriu_vector    */

void escriu_vector(estudiant t[LongTaula], int n) {
  int i;
  for (i=0; i<n; i++) escriu_estudiant(t[i]);
}
/* Acció: ordena_mat */
void ordena_mat(estudiant t[LongTaula], int n) {
   int i, j, posmin;
   for (i=0; i<n-1; i++) {
     posmin=i;
     for (j=i+1; j<n; j++) {
        if (menor_mat(t[j],t[posmin])) posmin=j;
     }
     intercanvia_estudiant(t[i],t[posmin]);
   }
}
/* Acció: intercanvia_estudiant */
void intercanvia_estudiant(estudiant& e1, estudiant& e2) {
    estudiant aux;
    copia_estudiant(aux,e1);
    copia_estudiant(e1,e2);
    copia_estudiant(e2,aux);
}
```

```
/* Funció: menor_mat */
bool menor_mat(const estudiant &e1, const estudiant &e2) {
  return (e1.nombre_matricula < e2.nombre_matricula);
}
```

6.7 Exercicis proposats

Aquesta secció proposa tota una sèrie d'exercicis per resoldre:

1

Definiu el tipus de dades *punt* que representen la coordenada x i la coordenada y d'un punt. Un cop fet això, implementeu un *main* que demani a l'usuari dos punts qualssevol i, mitjançant dues funcions o accions, responeu les preguntes següents:

- Quina és la distància entre tots dos punts? (Cal crear la funció *int distancia(punt a, punt b)*.)
- Aquests dos punts formen una recta? (*bool formen_recta(punt a, punt b)*.)

2

Definiu un tipus de dades complex que representi nombres complexos, és a dir, nombres amb una part real i una part imaginària. Cal fer un *main* que llegeixi una seqüència de nombres complexos acabada en $0+0i$ (és a dir, el complex amb part imaginària i real 0). En aquest programa, cal declarar les funcions/accions següents:

- *void llegeix_complex(complex &c);*
- *void escriu_complex(complex c)*, que escriu amb format: $4+5i$ o $3-2i$, segons el signe de la part imaginària.
- *bool es_zero(complex c)*, que retorna **cert** si el nombre complex és 0 (és a dir, és el sentinella) i **fals** altrament.
- *complex suma_complexos(complex c1, complex c2)*, que suma $c1$ i $c2$, i retorna un complex equivalent a la suma de tots dos.

3

Creeu un petit programa per gestionar una biblioteca. Cal definir l'estructura *llibre* que contingui: *autor, títol, ISBN, nombre de pàgines i editorial*. Un cop fet això, el programa ha de ser capaç de donar d'alta llibres i fer cerques per qualsevol dels seus camps. Com a suggeriment, es recomana:

- Declareu una taula de llibres. En aquesta taula, s'aniran guardant els llibres a la biblioteca quan es donin d'alta.
- Creeu les accions / funcions següents:
  - *void afegir_llibre(llibre v[TAMANY], llibre l)*, que afegeix el llibre l a la taula v si encara no és plena.
  - Creeu tantes funcions de consulta com vulgueu. Per exemple, *int cerca_llibre(llibre v[TAMANY], string nom_autor)*, que troba el primer llibre amb autor *nom_autor*, dins de la taula v.

  Fixeu-vos que, en aquest exercici, és important saber, en tot moment, fins a quina posició és plena la taula de llibres. No cal que implementeu persistència de dades.

4

Generalitzeu la proposta de l'exercici anterior per a qualsevol tipus de col·lecció: ja siguin CD de música, películes de vídeo, llibres, revistes, videojocs, etc. L'estructura a declarar dependrà del que vulgueu classificar. Les funcionalitats bàsiques del vostre programa seran les que s'han explicat a l'exercici anterior, però podeu ampliar l'exercici al vostre gust. Per exemple, donar de baixa elements de la col·lecció (penseu bé com fer-ho depenent de l'estructura de l'aplicació), consultar-los (segons altres criteris que se us ocorrin) o ordenar la vostra col·lecció per algun criteri (per exemple, per ordre alfabètic del nom d'autor). Si us interessa emprar aquest exercici en el vostre dia a dia, podeu implementar persistència de dades emprant fitxers (podeu fer un cop d'ull a: http://www.cplusplus.com/doc/tutorial/). L'estructura que heu d'emprar també la deixem a la vostra decisió; segurament, taules d'estructures serà una bona solució.

6.8 Conceptes avançats

Davant el creixement de l'enginyeria del software i l'increment de la complexitat dels programes desenvolupats, va sorgir la necessitat de garantir la qualitat del software (és a dir, del programari) desenvolupat. Amb aquesta finalitat es van proposar diferents *paradigmes* de programació. Un paradigma no és sinó un seguit de regles a mode de model que cal preservar per garantir la qualitat dels programes resultants. D'entre tots els paradigmes de programació existents, el més estès i de més impacte avui dia és la *programació orientada a objectes*.[1] També coneguda per les seves sigles *POO* (*OOP, Object Oriented Programming* en anglès).

La programació orientada a objectes pretén apropar el món real a la programació, és a dir, emular-la. La POO reprodueix la percepció de la realitat tal com la veien els antics filòsofs grecs, perfectament exemplificada al *mite de la caverna* de Plató. Per a ells, la realitat és una col·lecció d'objectes (o individus) que interaccionen entre si, i cada objecte pot veure's com un actor que interacciona (es comunica) amb la resta d'objectes de la realitat.

> El mite de la caverna és l'al·legoria més famosa de Plató, que es pot trobar al principi del setè llibre de la *República*; la seva obra més influent.

D'acord amb la programació orientada a objectes, qualsevol element conceptual per representar als nostres programes s'ha de representar com un objecte: que no és més que la instanciació d'una *classe*. Per exemple, la idea abstracta o concepte de cavall seria la classe, i *Rossinant* (el famós cavall de *Don Quixot de la Manxa*) seria un objecte o instanciació de la classe.

Les classes defineixen les qualitats (és a dir, les característiques pròpies dels individus) que els objectes o instàncies de la classe compartiran i els mecanismes amb què es comunicaran amb la resta d'individus o objectes. Per exemple, tots els cavalls tenen quatre potes, cua i crinera i, entre d'altres característiques, renillen, poden córrer a gran velocitat o trotar. En els termes d'aquest llibre i d'acord amb el que hem vist fins ara, la classe seria el tipus de dades i l'objecte, la declaració concreta d'una variable d'aquest tipus.

L'aportació d'aquest paradigma als nostres programes pot resumir-se com segueix: un alt grau de *modularitat* (que repercuteix en una major *reusabilitat* del codi), facilitat de manteniment i escalabilitat dels nostres programes, i una major elegància i comprensió del codi implementat, totes elles característiques desitjables si volem garantir la qualitat del programari.

Aprofundint una mica més, el concepte de classe no és sinó l'extensió del concepte de tupla. Una classe es compon d'*atributs* i *mètodes*. Els atributs són un conjunt de variables contingudes dins de la classe

[1] Aquesta secció del capítol és opcional i tan sols pretén introduir, breument, els conceptes de *classe* i *objecte*, presentats en aquest paradigma. Així doncs, l'objectiu d'aquesta secció és simplement introductori.

(exactament la mateixa idea que en el cas de les tuples i els seus camps o membres), mentre que els mètodes són prototipus de funcions i accions que manipulen la informació (atributs) continguda a la classe. Cal adonar-se, doncs, que els atributs guarden les qualitats que compartiran els objectes de la classe, mentre que els mètodes proporcionen als individus de la classe mecanismes per a comunicar-se.

Un exemple de declaració d'una classe podria ser el següent:

```
//Declarem la classe rectangle
class Rectangle {
    /* Dos variables senceres que
    representen la base i
    alçada del rectangle */
    int base, alcada;
  public:
    //Prototipus de les funcions i accions de la classe
    void fixar_base_i_alcada(int b, int a);
    int calcula_area();
};
```

La declaració dels atributs ***base*** i ***alcada***, igual que a les estructures, especifica les qualitats rellevants per al nostre programa de la classe implementada. En aquest cas, un rectangle es defineix per la seva base i la seva alçada. A més, també afegim el prototipus d'una funció (***int calcula_area()***) i una acció (***void fixar_base_i_alcada (int b,int a)***), que caldrà declarar més endavant. Aquests mètodes també estaran disponibles (és a dir, seran heretats) pels objectes que instanciïn aquesta classe. Per exemple, l'acció ***fixar_base_i_alcada*** ens permet fixar els valors de les variables ***base*** i ***alcada*** (els mètodes de la classe poden accedir als camps d'aquesta, encara que no es passin com a paràmetre; com si fossin una mena de *variables globals* dins de la classe). A més, la funció ***calcula_area()*** ens retornarà automàticament l'àrea del rectangle (consultarà les variables ***base*** i ***alcada*** de l'objecte en qüestió i en retornarà la seva multiplicació).

Un cop hem declarat un objecte (és a dir, una variable) de tipus ***rectangle*** (en el nostre exemple, de nom ***rect***), ja podem invocar (cridar) els seus mètodes. Per exemple, cridant correctament l'acció ***fixar_base_i_alcada*** podrem inicialitzar els valors dels atributs ***base*** i ***alcada***; posteriorment, cridant la funció ***calcula_area***, obtindrem el càlcul de l'àrea del rectangle ***rect***:

```
int main(void){
Rectangle rect;
 ...
rect.fixar_base_i_alcada(1,3);
cout << "L'àrea d'un rectangle de
base 1 i alçada 3 és "; cout << rect.calcula_area(); }
```

Els conceptes de classe i objecte són les pedres angulars d'aquest paradigma. No obstant això, n'hi ha d'altres conceptes igualment rellevants i necessaris: com l'*herència*, el *poliformisme*, els *templates* o les *excepcions*. En qualsevol cas, aquests conceptes van més enllà de l'objectiu d'aquesta secció, que és merament introductori. Per a més informació, es recomana consultar qualsevol tutorial o llibre de POO en C++.

7 Exemples d'aplicació

7.1 Exemples d'aplicacions amb fitxers d'àudio

Tots els fitxers multimèdia, tant si contenen àudio, com imatges o vídeos, en el fons són un conjunt de bits que representen els sons i els colors. Aquestes dades s'escriuen en formats diversos (*char*, *integer*, *float* ...), s'agrupen en tipus d'estructures de dades (*arrays*, *structs* ...) i s'organitzen de diferents maneres en els fitxers binaris (wav, jpg, mpg, avi ...). A més dels bits amb les mostres d'àudio i els píxels d'imatge, els fitxers contenen la informació necessària per poder-los reproduir o visualitzar adequadament.

A continuació, farem una explicació breu de com són els fitxers d'àudio, i proposarem uns quants projectes de programació per obrir, manipular i crear els nostres propis fitxers.

7.1.1 Els fitxers d'àudio

Els equips d'enregistrament d'àudio converteixen el so que arriba del micròfon en un senyal elèctric. A continuació, la targeta de so digitalitza l'amplitud del senyal en nombres binaris, que es guarden a la memòria o al disc de l'ordinador (v. figura 7.1). Per escoltar-los, se segueix el pas invers, és a dir, convertint els nombres binaris en valors d'amplituds, i aquests en un senyal elèctric que s'enviarà a l'altaveu a través d'un amplificador.

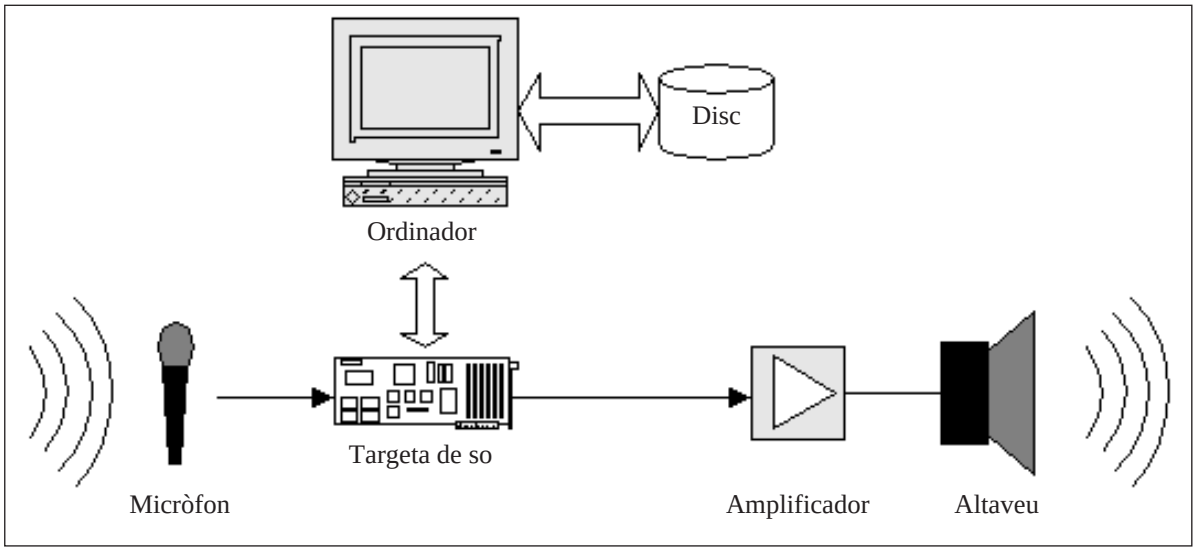


Fig. 7.1 Procés de digitalització d'àudio

La digitalització consisteix en la conversió de variables contínues, com pot ser un senyal elèctric d'àudio, en valors numèrics digitals. Per exemple, els fitxers extrets d'un CD-àudio contenen dos senyals (canals esquerre i dret), cadascun d'ells format per una seqüència de valors (mostres) de tipus *integer* de 16 bits, que representen l'amplitud del senyal i que estan agafats a una velocitat de 44.100 mostres per segon (freqüència de mostratge).

Els reproductors necessiten saber quins van ser els valors amb els quals es va digitalitzar el senyal per poder reproduir-lo de la mateixa manera. És per això que la majoria de fitxers incorporen aquesta informació en un bloc de dades al principi del fitxer, anomenat capçalera (*header*). Els fitxers d'àudio WAV contenen una capçalera molt simple i després totes les mostres sense comprimir. En alguns fitxers, com per exemple els populars MP3, també es poden incloure d'altres dades, com el títol i l'autor de l'obra (anomenades *metadades*).

Tots els fitxers digitals es poden comprimir per tal que ocupin menys. Això es fa mitjançant una transformació de les mostres d'àudio. Normalment, el procés de compressió introdueix uns petits errors entre els fitxers originals i els fitxers descomprimits, que es procura que no siguin perceptibles auditivament quan després es tornen a descomprimir per recuperar la música. Entre els fitxers comprimits més coneguts tenim els MP3, WMA, MPEG, etc.

7.1.2 Manipulació de fitxers d'àudio

Els fitxers WAV tenen una estructura molt ben definida on hi ha cadascuna de les informacions i de les mostres dels canals d'àudio. Necessitarem d'una banda, una estructura de dades que ens permeti manipular els senyals i, de l'altra, un parell de funcions per tal de llegir o escriure els fitxers WAV. Al fitxer AUDIOWAV.H teniu les definicions:

```
#ifndef AUDIOWAV_H
#define AUDIOWAV_H

namespace audioWav
{
// definicions de tipus
typedef struct {
  unsigned int canals;        //nombre de canals (stereo o mono)
  unsigned int frequencia;    //freqüència de mostratge (32000, 44100, etc)
  unsigned int num_mostres;   //nombre de mostres per canal
  short int** mostres;        //punter a les dades
} TWAV;

// funcions
int llegeix_wav(char* fitxer, TWAV* audio);
int escriu_wav(char* fitxer, TWAV* audio);

void inicialitza_wav(TWAV* audio, unsigned int canals, unsigned int num_mostres, unsigned
int frequencia);
void allibera_wav(TWAV* audio);

}
#endif
```

Vegem en detall l'estructura de dades que farem servir. Fixeu-vos que les mostres del senyal estan guardades en un punter doble de tipus *short int*. Això significa que el valor del camp *"mostres"* apunta el

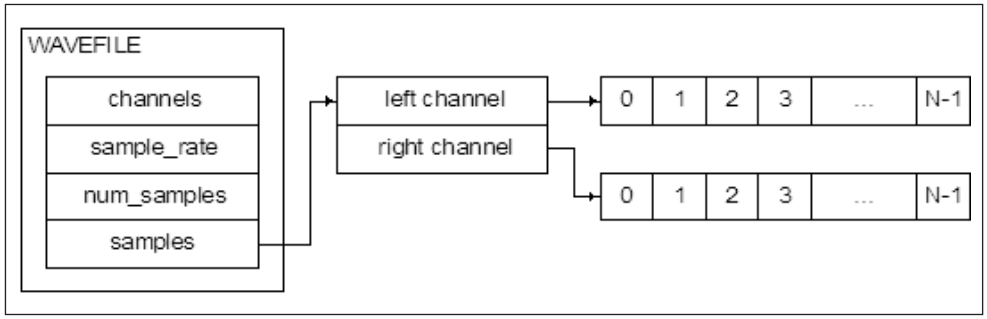


Fig. 7.2 Estructura de dades TWAV

principi d'un vector de punters ***short int***, on cadascun d'ells és un punter ***short int*** a la primera mostra (de tipus ***short int***) de cada canal d'àudio (v. figura 7.2).

La primera funció és òbviament per llegir un fitxer WAV i emmagatzemar-lo en una variable de tipus *TWAV*. A l'altra funció fem el pas contrari, és a dir, guardem una variable *TWAV* en un fitxer WAV. D'altra banda, la tercera funció serveix per inicialitzar una variable TWAV i reservar espai de memòria per posar-hi un senyal amb les característiques especificades, mentres que la quarta serveix per alliberar aquest espai de memòria. Per comprovar que les funcions fan el que nosaltres volem, i que som capaços de compilar un programa i executar-lo, farem el següent. A l'editor de text, obrirem un fitxer anomenat TESTWAV.CPP i hi introduirem el codi font següent:

```
#include <cstdlib>
#include <iostream>
#include "audioWAV.h"

using namespace std;
using namespace audioWav;

/* TESTWAVE
 * Prova de les funcions de lectura i escriptura
 */

int main()
{
   // Definicions de variables i constants
   const int MAXCAD = 40;
   char f1[MAXCAD+1], f2[MAXCAD+1];
   TWAV wave;

   // Introducció d'arguments
   cout << "Fitxer d'entrada: ";
   cin.getline(f1,MAXCAD+1,'\n');
   cout << "Fitxer de sortida: ";
   cin.getline(f2,MAXCAD+1,'\n');

   llegeix_wav(f1,&wave);
   escriu_wav(f2,&wave);

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

En el programa DevC++, compilarem aquest programa junt amb els fitxers AUDIOWAV.H i AUDIOWAV.CPP. Això ens ha de generar un fitxer executable anomenat TESTWAV. Bàsicament, aquest programa el que fa és llegir un fitxer WAV d'entrada, carregar-lo en la variable WAVE i escriure un nou fitxer WAV de sortida utilitzant la mateixa variable. Per copiar d'una variable a l'altra, fixeu-vos que no cal copiar totes les mostres sinó simplement indicar que el punter del senyal de sortida ha d'apuntar les mostres (i els canals) a la mateixa adreça que per al senyal d'entrada.

Si l'executem amb un fitxer WAV qualsevol, els senyals d'entrada i de sortida haurien de ser exactament iguals. Podeu comprovar-ho reproduint-lo o, encara millor, obrint-lo en un editor d'àudio (per exemple: Audacity, Wavesurfer, etc.).

Ara que ja coneixem com són les dades i les funcions bàsiques per a obrir i manipular fitxers d'àudio, podem començar a programar.

Programa CAPGIRA.CPP

El nostre primer programa consistirà a obrir un fitxer WAV, que retornarem girat de l'inrevés (v. figura 7.3). En aquest projecte, un cop llegit el fitxer WAV original, haurem d'inicialitzar una nova variable de tipus TWAV per poder-hi guardar les mostres del senyal girat. Per a cadascun dels canals, haurem de copiar les mostres d'un senyal a l'altre tenint en compte que la mostra 0 ara serà la mostra $N-1$, la mostra 1 serà la $N-2$, i així successivament fins a la mostra final, que serà la primera mostra del senyal girat (N és el nombre de mostres del senyal).

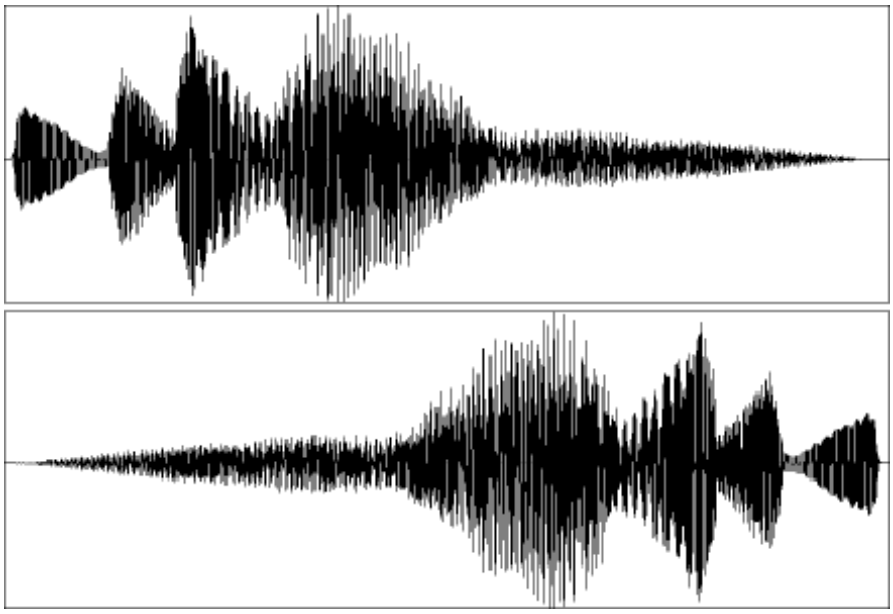


Fig. 7.3 Fitxer original i fitxer invertit en el temps

De la mateixa manera que abans, haurem de compilar el programa i executar-lo.

```
#include <cstdlib>
#include <iostream>
#include "audioWav.h"

using namespace std;
using namespace audioWav;
```

```
/* REVERSE
 * Capgira un fitxer d'àudio
 *        reverse.exe    in_file.wav    out_file.wav
 */

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

  TWAV waveIn, waveOut;
  unsigned int n;
  int c;

  if(argc < 3){
 cout << "ERROR: és necessari passar-li dos arguments" << endl;
     cout << "Exemple: reverse.exe  infile.wav  outfile.wav" << endl;
     system("PAUSE");
 return EXIT_FAILURE;
  }

  // Llegeix fitxer d'entrada
  cout << "Llegint " << argv[1] << endl;
  llegeix_wav(argv[1], &waveIn);

  inicialitza_wav(&waveOut,waveIn.canals,waveIn.num_mostres,waveIn.frequencia);

  for (c=0;c<waveIn.canals;c++) {
    for (n=0;n<waveIn.num_mostres;n++) {
 waveOut.mostres[c][waveIn.num_mostres-1-n] = waveIn.mostres[c][n];
    }
  }

  // Escriu fitxer de sortida
  cout << "Escrivint " << argv[2] << endl;
  escriu_wav(argv[2], &waveOut);

  // Allibera memòria
  allibera_wav(&waveIn);
  allibera_wav(&waveOut);

  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Programa LA440.CPP

A continuació, fem un programa per generar un so corresponent a la nota La a 440 Hz, que és la que serveix per afinar els instruments.

A diferència dels projectes anteriors, no llegim cap fitxer d'entrada sinó que generem un nou fitxer des del no-res. Serà un fitxer monofònic. Fixeu-vos que quan inicialitzem la variable del senyal definim un únic canal, i quan assignem els valors de les mostres ho fem sobre el $canal = 0$. Igualment, necessitarem definir unes quantes variables amb les característiques del so que volem generar.

```
#include <cstdlib>
#include <iostream>
#include <math.h>
#include "audioWAV.h"

using namespace std;
using namespace audioWav;

#define PI 3.141592

/* La440
 * Generació d'un to sinusoidal a 440Hz
 *        la440.exe    outfile.wav
 */


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

  TWAV waveOut;

  int f0 = 440;   // freqüència del to (Hz)
  int fm = 44100;   // freqüència mostratge (Hz)
  int durada = 5;    // (segons)
  double amplitud = 0.8; // escalat a 1

  unsigned int n;
  unsigned int num_mostres = fm*durada;

  if(argc < 2){
 cout << "ERROR: és necessari passar-li un argument" << endl;
    cout << "Exemple: la440.exe  outfile.wav" << endl;
    system("PAUSE");
 return EXIT_FAILURE;
  }


  amplitud = amplitud*32768; // escalat a 16-bits

  inicialitza_wav(&waveOut,1,num_mostres,fm);

  for (n=0;n<num_mostres;n++) {
 waveOut.mostres[0][n] = (short int)(amplitud*sin((double)(2*PI*f0*n/fm)));
  }

// Escriu fitxer de sortida
  cout << "Escrivint " << argv[1] << endl;
  escriu_wav(argv[1], &waveOut);

  allibera_wav(&waveOut);

  system("PAUSE");
  return EXIT_SUCCESS;
}
```

La funció que ens dóna les mostres del senyal és la funció sinusoïdal

$$x[n] = A * \sin (2 * pi * F * n / Fm)$$

Per poder treballar correctament amb la funció sinus, caldrà enllaçar el codi del programa amb la llibreria de funcions matemàtiques i treballar amb un tipus de dades *double* per a tenir millor precisió.

Programa SIRENA.CPP

A continuació, fem un programa per generar un so de sirena. Aquest so consistirà en una repetició de tres tons curts de freqüències 400 Hz, 600 Hz i 400 Hz, separats per una pausa entre tons de 50 ms, i un silenci entre cada grup de tres tons de 500 ms. La durada de cada to ha de ser de 100 ms. L'amplitud del senyal és 0,8 i la seva freqüència de mostratge, 44.100 Hz.

El senyal s'ha de veure més o menys:

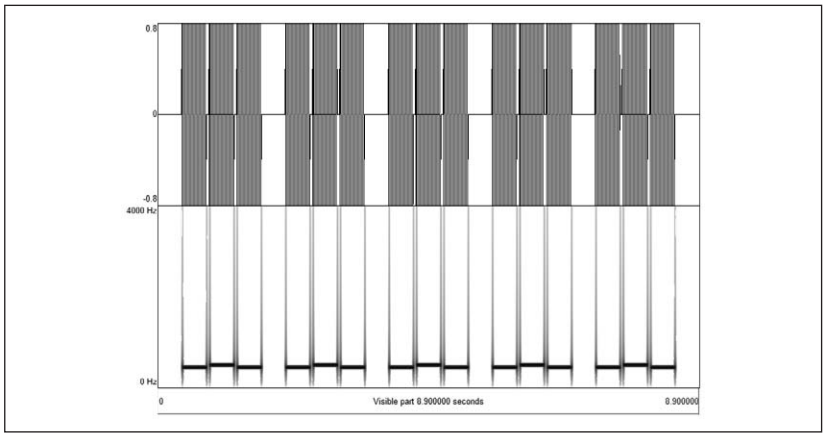


Fig. 7.4 Forma d'ona i espectrograma del senyal amb la sirena de tres tons (segons el programa de visualització i dels paràmetres de configuració es pot veure diferent)

```
#include <iostream>
#include <cstdlib>
#include <math.h>
#include "audioWAV.h"

using namespace std;
using namespace audioWav;

#define PI 3.141592

/* SIREN
 * Genera un senyal que conté un to de sirena alternant dos tons
 *        siren.exe      outfile.wav
 */

// Funció principal
int main(int argc, char **argv) {
  TWAV waveOut;
```

```
 unsigned int f1 = 450; // freqüència baixa (Hz)
 unsigned int f2 = 500; // freqüència alta (Hz)
 unsigned int fm = 44100; // freqüència mostratge (Hz)
 double dur = 0.4;    // durada del to (segons)
 double pau1 = 0.05;  // pausa entre tons (segons)
 double pau2 = 0.4;   // pausa entre repeticions (segons)
 int rep = 5;     // número de repeticions

 double amplitud = 0.8; // amplitud escalada a 1
 short int amplitud = int(amplitud * (double)32768); // escalada a 16-bits

 int r;

 if(argc < 2){
cout << "ERROR: és necessari passar-li un argument" << endl;
   cout << "Exemple: siren.exe  outfile.wav" << endl;
   system("PAUSE");
return EXIT_FAILURE;
 }

 // durada total
 double durtotal = pau2*(rep+1) + rep*(3*dur + 2*pau1);
 unsigned int num_mostres = (unsigned int)(fm*durtotal);

 inicialitza_wav(&waveOut,1,num_mostres,fm);

 short int *punter;

 p_sample = waveOut.mostres[0];

 // starting pause
 num_mostres = tone(punter, 0, 0, pau2, sampling);
 punter += num_mostres;

 for (r=0;r<rep;r++) {
   num_samples = tone(punter, amplitud, freq1, dur, fm);
   punter += num_mostres;

   num_samples = tone(punter, 0, 0, pau1, fm);
   punter += num_mostres;

   num_samples = tone(punter, amplitud, freq2, dur, fm);
   punter += num_mostres;

   num_samples = tone(punter, 0, 0, pau1, sampling);
   punter += num_mostres;

   num_samples = tone(punter, amplitud, freq1, dur, fm);
   punter += num_mostres;

   num_samples = tone(punter, 0, 0, pau2, fm);
   punter += num_mostres;
 }
```

```
  // Escriu fitxer de sortida
  cout << "Escrivint " << argv[1] << endl;
  escriu_wav(argv[1], &waveOut);

  // Allibera memòria
  allibera_wav(&waveOut);

  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Hem definit una funció dins aquest programa per a realitzar la generació d'un to d'una determinada freqüència i durada perquè és una acció que es repeteix diverses vegades dins el codi del programa principal.

```
//   Funció auxiliar per genenar les mostres d'un to sinusoidal
int tone(short int* mostres, unsigned int amplitud, unsigned int f0, double durada,
unsigned int fm)
{
  unsigned int n;
  unsigned int num_mostres;

  num_mostres = (unsigned int)(fm * durada);

  for (n=0;n<num_mostres;n++) {
 mostres[n] = (short int)(amplitud*sin((double)(2*PI*f0*n/fm)));
  }
  return(num_mostres);
}
```

Proveu de canviar les característiques del senyal (freqüències, durades, pauses, nombre de repeticions, etc.).

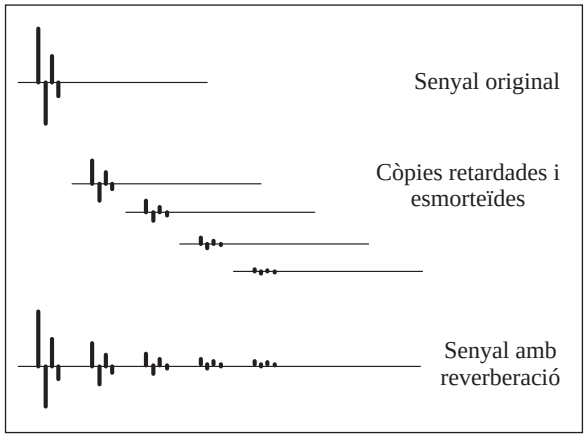


Fig. 7.5 Generació de la reverberació com a suma d'ecos

Programa REVERB.CPP

Finalment, fem un programa per afegir reverberació de forma digital a un senyal d'àudio. El principi de funcionament de la reverberació és crear còpies del propi senyal d'entrada, que es van sumant a ell mateix amb unes amplituds cada vegada més petites. És a dir, com un eco que es va perdent (v. figura 7.5).

El programa llegeix el fitxer d'entrada. Posteriorment, converteix el retard en segons (o mil·lisegons) a mostres, calcula la longitud total del nou senyal comptant les repeticions que caldrà afegir-hi i inicialitza una nova variable TWAV per al senyal de sortida.

```
#include <cstdlib>
#include <iostream>
#include "audioWAV.h"

using namespace std;
using namespace audioWav;


/* Reverberació
 *      reverb.exe      file_in.wav      file_out.wav
 */

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

  TWAV waveIn, waveOut;

  unsigned int retard = 500;          /* (milisegons) */
  double guany = 0.95;                /* guany del retard; ha de ser menor que 1 !!! */
  int  ntaps = 10;                    /* número d'ecos */
  unsigned int total_mostres;

   if(argc < 3){
 cout << "ERROR: és necessari passar-li dos arguments" << endl;
 cout << "Exemple: reverb.exe  infile.wav  outfile.wav" << endl;
     system("PAUSE");
 return EXIT_FAILURE;
  }

  /* Llegeix fitxer d'entrada */
  cout << "Llegint " << argv[1] << endl;
  llegeix_wav(argv[1], &waveIn);

  retard = (unsigned int)((double)retard/1000*(double)waveIn.frequencia);
  total_mostres = waveIn.num_mostres + ntaps*retard;
  inicialitza_wav(&waveOut,waveIn.canals,total_mostres,waveIn.frequencia);

  unsigned int n;
  int c;
  int tap;
  double gtap;

  for (c=0;c<waveIn.canals;c++) {

     /* copia senyal original */
     for (n=0;n<waveIn.num_mostres;n++) {
    waveOut.mostres[c][n] = waveIn.mostres[c][n];
     }
     /* afegeix zeros fins al final */
     for (n=waveIn.num_mostres; n<total_mostres;n++) {
    waveOut.mostres[c][n] = 0;
     }

     /* afegeix senyal retardat */
```

```
      gtap = guany;
      for (tap=1;tap<=ntaps;tap++) {
        for (n=0;n<waveIn.num_mostres;n++) {
          waveOut.mostres[c][n+tap*retard] += (short int)(gtap*(double)waveIn.mostres[c][n]
        }
        gtap = gtap*gtap; /* cada vegada el guany es redueix */
      }
    }

    /* Escriu fitxer de sortida */
    cout << "Escrivint " << argv[2] << endl;
    escriu_wav(argv[2], &waveOut);

    /* Allibera memòria */
    allibera_wav(&waveIn);
    allibera_wav(&waveOut);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Per generar el senyal, agafa les mostres del senyal d'entrada i les copia sobre el senyal de sortida, i ho completa amb zeros per assegurar que no hi ha res. Després, fa un bucle per a cada repetició (eco) i en copia les mostres originals, multiplicades (o, més ben dit, disminuïdes) pel factor de guany, tenint en compte el desplaçament del retard.

7.2 Exemples d'aplicació amb fitxers d'imatge

De la mateixa manera que els fitxers d'àudio, els fitxers que contenen imatges són també un conjunt de bits que representen punt a punt la informació d'una imatge. Aquesta informació es refereix tant a la imatge pròpiament dita com a tot el que és necessari per poder-la visualitzar adequadament.

7.2.1 Els fitxers d'imatge. El format BMP

En un fitxer d'imatge, hi ha d'haver tota la informació necessària per poder tractar la imatge que conté. Normalment, se'n poden diferenciar dues parts principals:

- *Capçalera del fitxer:* amb informació essencial sobre el contingut del fitxer i les característiques de la imatge: dimensions, codificació, paleta (taula descriptiva) de colors o blanc i negre, compressió.
- *Dades de la imatge:* informació sobre la representació de la imatge punt a punt segons l'organització definida a la capçalera.

Segons quina sigui la codificació d'aquestes parts, es poden obtenir els diferents tipus de formats de fitxers d'imatge: BMP, JPEG, GIF, etc. En aquesta secció, es tracta el format BMP (bitmap file format) ja que és un dels més senzills i utilitzats malgrat l'inconvenient de la grandària dels seus arxius. És l'estàndard de Microsoft i, per això, el format propi de Microsoft Paint. Un arxiu BMP és un mapa de bits de manera que els píxels es troben en forma de taula de punts on els colors poden ser reals (*Truecolor*) o donats sobre la base d'una paleta gràfica que els descriu. De fet, hi ha diferents tipus de mapes de bits segons si la imatge és en blanc i negre, de 16 colors, de 256 colors, *Hicolor* o *Truecolor*, ja que per a cada cas es farà servir un determinat nombre de bits (v. taula 7.1).

Tipus de color	Descripció
B/N	1 bit per píxel (1 byte conté 8 píxels).
16	4 bits per píxel (1 byte conté 2 píxels).
256	8 bits per píxel (1 byte conté 1 píxel).
Hicolor	16 bits per píxel (2 bytes contenen 1 píxel).
Truecolor (16,7 milions)	24 bits per píxel (4 bytes contenen 1 píxel).

Taula 7.1 Codificació dels colors

Un arxiu BMP s'estructura en quatre parts, cadascuna amb la seva mida i funcionalitat:

Capçalera del fitxer Conté informació relativa al tipus de mapa, per exemple un mapa de bits estàndard comença amb les lletres 'BM' (en hexadecimal: 0x42 0x4D), grandària de l'arxiu, i on comença la descripció píxel a píxel de la imatge. Ocupa 14 bits.

Capçalera d'informació Conté les característiques generals de la imatge, com són la seva grandària i el nombre de colors, amb la seva paleta de colors quan aquesta és necessària, entre d'altres informacions. Ocupa 40 bytes (v. taula 7.2).

Mida	Descripció
4 bytes	Mida en bytes de la capçalera d'informació. Sempre és 40.
4 bytes	Amplada en píxels de la imatge.
4 bytes	Alçada en píxels de la imatge.
2 bytes	Plans. Sense ús.
2 bytes	Profunditat de color. (Bits per píxel: 8 o 24)
4 bytes	Tipus de compressió. En els BMP és zero.
4 bytes	Mida de l'estructura de dades
4 bytes	Píxels per metre horitzontal. Sense ús.
4 bytes	Píxels per metre vertical. Sense ús.
4 bytes	Nombre de colors usats. Sense ús.
4 bytes	Nombre de colors "importants". Sense ús.

Taula 7.2 Capçalera d'informació

Paleta de colors Només és necessària per a imatges de 4 i 8 bits i, per tant, la seva incorporació és opcional. Per exemple, en el cas de 8 bits, hi ha 256 colors possibles, codificats des del 0 fins al 255, segons una paleta (taula) que fixa la correspondència de cada color amb el seu número, mitjançant la barreja dels colors: vermell (R), verd (G) i blau (B). En el cas d'imatges de 24 bits, la paleta és innecessària i no es carregarà.

Dades de la imatge Informació de la imatge pròpiament dita píxel a píxel, definida d'esquerra a dreta i des de la línia inferior cap a la superior.

7.2.2 Exemple: lectura d'un fitxer BMP

Primerament, es presenta un programa en C++ per llegir un arxiu BMP. Per poder llegir la imatge continguda en un arxiu BMP, s'ha vist que hi ha molts bytes que no són útils però s'han de llegir tots fins arribar a la imatge. En total, sempre haurem de llegir 54 bytes, que es correspondran amb les dues capçaleres (d'arxiu i d'informació), i per poder accedir a la seva informació es disposa de les variables:

```
typedef byte TCap_Arxiu[14];

typedef struct {
    int dimcabinfo;
    int ample;
    int alt;
    word plans;
    word bits;
    int comp;
    int paleta_pixel;
    int ppm_h;
    int ppm_v;
    int colors;
    int colors_imp;
}TCap_Informacio;
```

La primera es correspon amb la capçalera d'arxiu i la segona, amb la capçalera d'informació, on els tipus *byte* i *word* s'han de crear prèviament com a tipus de dada:

```
typedef unsigned char byte;
typedef unsigned short word;
```

En llegir la capçalera d'informació *TCap_Informacio*, es disposa d'un camp (*bits*) que informa sobre el nombre de bits de la imatge; si és 8, s'haurà de carregar la paleta i, si és 24, no s'haurà de carregar. En cas que es carregui la paleta, consistirà en els valors RGB (*red, green, blue*) per a cada color (256) i un byte zero entre cada terna de valors, per la qual cosa s'ha de definir una taula de 256 tuples on cada tupla ha de definir cadascun dels colors:

```
typedef struct {
    byte b;
    byte g;
    byte r;
    byte separador;
}Trgb;
typedef Trgb TPaleta[NUM_COLORS];
```

Finalment, la descripció de les dades de la imatge píxel a píxel es pot aconseguir a partir d'una taula, per exemple de tipus de dades *byte* si es volen processar imatges de 8 bits/píxel (de tipus *word* serà per a imatges de 16 bits/píxel i de tipus *int* per a imatges de 32 bits/píxel). En el programa presentat, es fa per 8 bits (1 byte):

```
typedef byte TDades[MAX_AMP][MAX_ALT];
```

D'aquesta manera, una imatge sencera ve donada a partir de la tupla:

```
typedef struct {
    TCap_Arxiu cap_arxiu;
    TCap_Informacio cap_informacio;
    TPaleta paleta;
    TDades dades;
}Tbmp;
```

Fig. 7.6 Lena (imatge de proves)

Programa LLEGIR_IMATGE.CPP

A continuació, es mostra el codi font d'un programa en C++ que permet llegir les característiques principals d'un arxiu d'imatge BMP de 8 bits. S'ha de tenir en compte el que s'ha vist a la secció 7.2.2 i, a més a més, el fet que el fitxer que conté la imatge (per exemple, lena.bmp, v. figura 7.6) s'ha de llegir com a fitxer binari, tal com es detalla al codi.

L'execució d'aquest programa per la imatge *lena.bmp* proporcionarà la sortida de la figura 7.7.

```
Escriu el nom del fitxer amb la imatge original (amb .bmp): lena.bmp
Imatge carregada

                INFORMACIO DE LA IMATGE CARREGADA

                  Informacio de l'arxiu: BM8♦♦     6♦
                  Tamany cap. informacio: 40
                  Amplada: 512
                  Alçada: 512
                  Bits-pixel: 8
                  Compresio: 0
                  Paleta i pixels: 0

Presione una tecla para continuar . . .
```

Fig. 7.7 Execució del programa per l'arxiu lena.bmp

```
#include <iostream>
#include <fstream>
using namespace std;

const int NUM_COLORS = 256;     //nombre colors per 8 bits de resolució
const int MAX_AMP=1024;         //amplada màxima de la imatge
```

```
const int MAX_ALT=768;          //alçada màxima de la imatge
const int MAX_CAD = 25;         //nombre màxim caràcter pel nom dels fitxers

typedef char nomfitxer[MAX_CAD];    //taula per desar el nom d'un fitxer
typedef unsigned char byte;         //1 byte són 8 bits (1 char sense signe)
typedef unsigned short word;        //1 word són 16 bits (integer curt sense signe)
typedef byte TCap_Arxiu[14]; //capçalera de l'arxiu de 14 bytes de mida
//Matriu per definir imatge pixel a pixel amb 8 bits per pixel
typedef byte TDades[MAX_AMP][MAX_ALT];

//Camps de la capçalera de informació
typedef struct {
   int dimcabinfo;
   int ample;
   int alt;
   word plans;
   word bits;
   int comp;
   int paleta_pixel;
   int ppm_h;
   int ppm_v;
   int colors;
   int colors_imp;
}TCap_Informacio;

//color en base a RGB
typedef struct {
    byte b;
    byte g;
    byte r;
    byte separador;
}Trgb;
//Paleta de color pels 256 colors possibles(taula de 256 tuples Trgb)
typedef Trgb TPaleta[NUM_COLORS];

//Tupla per definir una imatge en format BMP
typedef struct {
   TCap_Arxiu cap_arxiu;
   TCap_Informacio cap_informacio;
   TPaleta paleta;
   TDades dades;
}Tbmp;

void Llegir_imatge(nomfitxer nom, Tbmp &imatge, bool &ok);
void informacio(Tbmp imatge); // dades de la imatge llegida

int main()
{
  string s1;
  Tbmp imatge;
  bool ok;
  int i;
  nomfitxer nom1;
```

```
  cout << "Escriu el nom del fitxer amb la imatge original (amb .bmp): ";
  getline(cin,s1);
  //Passem el nom de tipus string a tipus char, ja que per manipular el fitxer
  //és millor així
  for(i=0;i<s1.size()+1;i++) nom1[i]=s1[i];
  Llegir_imatge(nom1,imatge,ok);
  if(ok) cout << "Imatge carregada" << endl;
  else cout << "Imatge no carregada" << endl;
  informacio(imatge);
  system("pause");
  return 0;
}


void Llegir_imatge(nomfitxer nom, Tbmp &imatge, bool &ok)
{
    //obrim el fitxer "nom" en mode binari amb la variable f_ent
    ifstream f_ent(nom, ios::binary);
    int x,y;
    ok=((f_ent.good()) && (!f_ent.fail())); //mirem si tot va bé
    if (ok)
    {
        //Llegim la capçalera de l'arxiu
        f_ent.read((char *)&(imatge.cap_arxiu),sizeof(TCap_Arxiu));
        //Llegim la capçalera amb informació
        f_ent.read((char *)&(imatge.cap_informacio),sizeof(TCap_Informacio));
        //Llegim la paleta de colors
        f_ent.read((char *)&(imatge.paleta),sizeof(TPaleta));
          //Llegim la imatge amb les seves dimensions reals
        for(x=0;x<imatge.cap_informacio.ample;++x)
        {
            for(y=0;y<imatge.cap_informacio.alt;++y)
            {
                f_ent.read((char *)&(imatge.dades[x][y]),sizeof(byte));
            }
        }
        f_ent.close(); //tanquem el fitxer
    }
}

void informacio(Tbmp imatge)
{
     int i;
     cout << endl << endl;
     cout << "                INFORMACIO DE LA IMATGE CARREGADA" << endl;
     cout << endl;
     cout<<"                    Informacio de l'arxiu: ";
     for (i=0; i<14; i++) cout << imatge.cap_arxiu [i];
     cout << endl;
     cout<<"                    Tamany cap. informacio: ";
     cout << imatge.cap_informacio.dimcabinfo << endl;
     cout<<"                    Amplada: "<<imatge.cap_informacio.ample<<endl;
     cout<<"                    Alçada: "<<imatge.cap_informacio.alt<<endl;
```

```
    cout<<"                    Bits-pixel: "<<imatge.cap_informacio.bits<<endl;
    cout<<"                    Compresio: "<<imatge.cap_informacio.comp<<endl;
    cout<<"                    Paleta i pixels: ";
    cout << imatge.cap_informacio.paleta_pixel<<endl;
    cout << endl;
}
```

7.2.3 Exemple: processat d'una imatge

Un cop llegit un fitxer amb una imatge, es pot processar la imatge a partir de les dades llegides i que omplen la taula *Tdades*. Si, per exemple, es vol posar un marc a la imatge, un cop triada la mida del gruix del marc (constant GRUIX al programa) i el seu color (constant COLOR al programa), s'hauran de col·locar, a les coordenades corresponents de la taula *Tdades*, els punts que formaran el marc que es vol col·locar. L'escriptura de la nova imatge (amb el marc ja posat) s'ha de fer de manera anàloga a la lectura del fitxer, però en aquest cas amb operacions de sortida.

Programa MARC.CPP

Programa en C++ que, un cop llegida una imatge continguda en un fitxer amb format BMP, li posa un marc d'un determinat gruix i color i la deixa en un fitxer també amb format BMP i amb el nom que li hagi proporcionat l'usuari. La imatge llegida ha de ser de 8 bits de profunditat de color.

```
#include <iostream>
#include <fstream>
using namespace std;

const unsigned char  COLOR=0;  //color del marc a posar 0-negre...255-blanc
const int GRUIX=15;            //gruix del marc a posar
const int DIM_CAP = 14;        //14 bytes tamany capçalera
const int NUM_COLORS = 256;    //nombre colors per 8 bits de resolució
const int MAX_AMP=1024;        //amplada màxima de la imatge
const int MAX_ALT=768;         //alçada màxima de la imatge
const int MAX_CAD = 25;        //nombre màxim caràcter pel nom dels fitxers

typedef char nomfitxer[MAX_CAD];    //taula per desar el nom d'un fitxer
typedef unsigned char byte;         //1 byte són 8 bits (1 char sense signe)
typedef unsigned short word;        //1 word són 16 bits (integer curt sense signe)
typedef byte TCap_Arxiu[DIM_CAP];   //capçalera de l'arxiu de 14 bytes de mida
//Matriu per definir imatge pixel a pixel amb 8 bits per pixel
typedef byte TDades[MAX_AMP][MAX_ALT];

//Camps de la capçalera d'informació
typedef struct {
   int dimcabinfo;
   int ample;
   int alt;
   word plans;
   word bits;
   int comp;
   int t_imagen;
   int ppm_h;
```

```
   int ppm_v;
   int colors;
   int colors_imp;
}TCap_Informacio;

//color en base a RGB
typedef struct {
    byte b;
    byte g;
    byte r;
    byte separador;
}Trgb;
//Paleta de color pels 256 colors possibles(taula de 256 tuples Trgb)
typedef Trgb TPaleta[NUM_COLORS];

//Tupla per definir una imatge en format BMP
typedef struct {
   TCap_Arxiu cap_arxiu;
   TCap_Informacio cap_informacio;
   TPaleta paleta;
   TDades dades;
}Tbmp;


void Llegir_imatge(nomfitxer nom, Tbmp &imatge, bool &ok);
void Escriure_imatge(nomfitxer nom, Tbmp &imatge, bool &ok);
void marc(Tbmp &imatge); // posa un marc a la imatge

int main()
{
  string s1,s2;
  Tbmp imatge;
  bool ok;
  int i;
  nomfitxer nom1,nom2;
  cout << "Escriu el nom del fitxer amb la imatge original (amb .bmp): ";
  getline(cin,s1);
  cout << "Escriu el nom del fitxer per desar la imatge final (amb .bmp): ";
  getline(cin,s2);
  //Passem el nom de tipus string a tipus char, ja que per manipular el fitxer
  //és millor així
  for(i=0;i<s1.size()+1;i++) nom1[i]=s1[i];
  Llegir_imatge(nom1,imatge,ok);
  if(ok) cout << "Imatge carregada" << endl;
  else cout << "Imatge no carregada" << endl;
  marc(imatge);
  for(i=0;i<s2.size()+1;i++) nom2[i]=s2[i];
  Escriure_imatge(nom2,imatge,ok);
  if(ok) cout << "Imatge carregada" << endl;
  else cout << "Imatge no carregada" << endl;

  system("pause");
  return 0;
```

```
}

//Posa un marc a la imatge de gruix GRUIX i color COLOR
void marc(Tbmp &imatge)
{
    int x,y;
    //marc lateral dreta
    for(y=0;y<GRUIX;y++)
       for(x=0;x<imatge.cap_informacio.ample;x++)
          imatge.dades[x][y]=COLOR;
    //marc lateral esquerra
    for(y=imatge.cap_informacio.alt-GRUIX;y<imatge.cap_informacio.alt;y++)
       for(x=0;x<imatge.cap_informacio.ample;x++)
          imatge.dades[x][y]=COLOR;
    //marc superior
    for(x=0;x<GRUIX;x++)
       for(y=0;y<imatge.cap_informacio.alt;y++)
          imatge.dades[x][y]=COLOR;
   //marc inferior
    for(x=imatge.cap_informacio.ample-GRUIX;x<imatge.cap_informacio.ample;x++)
       for(y=0;y<imatge.cap_informacio.alt;y++)
          imatge.dades[x][y]=COLOR;
}

void Llegir_imatge(nomfitxer nom, Tbmp &imatge, bool &ok)
{
    //obrim el fitxer "nom" en mode binari amb la variable f_ent
    ifstream f_ent(nom, ios::binary);
    int x,y;
    ok=((f_ent.good()) && (!f_ent.fail())); //mirem si tot va be
    if (ok)
    {
        //Llegim la capçalera de l'arxiu
        f_ent.read((char *)&(imatge.cap_arxiu),sizeof(TCap_Arxiu));
        //Llegim la capçalera amb informació
        f_ent.read((char *)&(imatge.cap_informacio),sizeof(TCap_Informacio));
        //Llegim la paleta de colors
        f_ent.read((char *)&(imatge.paleta),sizeof(TPaleta));
         //Llegim la imatge amb les seves dimensions reals
        for(x=0;x<imatge.cap_informacio.ample;++x)
        {
            for(y=0;y<imatge.cap_informacio.alt;++y)
            {
                f_ent.read((char *)&(imatge.dades[x][y]),sizeof(byte));
            }
        }
        f_ent.close(); //tanquem el fitxer
    }
}

void Escriure_imatge(nomfitxer nom, Tbmp &imatge, bool &ok)
{
    ofstream f_sort(nom, ios::binary);
```

```
    int x,y;
    ok=((f_sort.good()) && (!f_sort.fail()));
    if (ok)
    {
        f_sort.write((char *)&(imatge.cap_arxiu),sizeof(TCap_Arxiu));
        f_sort.write((char *)&(imatge.cap_informacio),sizeof(TCap_Informacio));
        f_sort.write((char *)&(imatge.paleta),sizeof(TPaleta));

        for(x=0;x<imatge.cap_informacio.ample;++x)
        {
            for(y=0;y<imatge.cap_informacio.alt;++y)
            {
                    f_sort.write((char *)&(imatge.dades[x][y]),sizeof(byte));
            }
        }
        f_sort.close();
    }
}
```

El resultat obtingut es mostra a la figura 7.8.

Fig. 7.8 Colocació d'un marc a la imatge

7.3 Exemples d'aplicació amb targetes d'entrada i sortida

Una de les aplicacions més importants de programació per als enginyers és en l'àmbit del control industrial. Mitjançant un programa realitzat en C++, podem controlar maquinari de diferents índoles. En aquesta secció, intentem exposar un exemple senzill de control de dispositius externs mitjançant un programa realitzat en C++.

7.3.1 Conceptes i elements bàsics

En el control de dispositius externs mitjançant un programa en C++, hem de tenir en compte els conceptes i elements següents:

Els ports i la connexió a l'ordinador. Per enviar i rebre dades d'uns dispositius a un ordinador, podem utilitzar els *ports*, que són interfícies de connexió amb l'exterior com vies d'entrada i sortida. Entre els ports accessibles sense haver d'obrir l'ordinador, podem destacar el port USB *(universal serial bus)*, que actualment és el més utilitzat, té una velocitat de transmissió acceptable i permet utilitzar el mateix connector per alimentar alguns dispositius externs, mentre que els ports sèrie (el més comú és el tipus RS-232) i el port paral·lel (també anomenat CENTRONICS), malgrat que van ser estàndards durant molts anys, actualment estan obsolets. Cal destacar el port PCMCIA (PC card) per connectar dispositius externs a un ordinador portàtil amb un bon rendiment i els ports sense fils, per la seva simplificació de hardware i per l'augment de l'accessibilitat.

Tipus de senyals en l'intercanvi d'informació. Normalment, existeixen dos tipus de senyals que un dispositiu extern pot rebre o enviar:

- Senyal digital: està format per un conjunt d'estats zeros i uns com a conseqüència de dos nivells de tensió elèctrica, alt (*high*) i baix (*low*). Per exemple: un polsador, un interruptor i un sensor digital generen una informació discreta o sencera (estat ON o OFF).
- Senyal analògic: quan la informació en el temps pot tenir una variació contínua. Per exemple, un sensor de temperatura origina una quantitat de voltatge en proporció amb la temperatura, que pot ser molt variada i no sols 1 i 0.

Conversió d'informació. L'ordinador és un sistema digital i, per tant, en el cas de la informació analògica, aquesta s'ha de convertir a digital per ser processada. El conversor analògic/digital (A/D) s'encarrega de convertir la informació analògica en una *dada digital* que pugui ser enviada a l'ordinador, i el conversor digital/analògic (D/A) fa una conversió inversa de la informació digital procedent de l'ordinador al dispositiu analògic.

Sensor i condicionador de senyal. El sensor és un element que té la capacitat de transformar una magnitud física, com per exemple la temperatura, en un senyal elèctric. Els sensors analògics, moltes vegades no poden generar una informació analògica entenedora per un conversor A/D. Per a això s'ha d'utilitzar un sistema condicionador de senyal (*signal conditioning*), que s'encarrega de treure el soroll, amplificar i adequar el senyal.

Circuit de potència. Per activar un dispositiu extern, normalment el senyal digital procedent de l'ordinador no té suficient força (potència elèctrica) i, per tant, s'han d'utilitzar circuits de potència que reforcin la seva força.

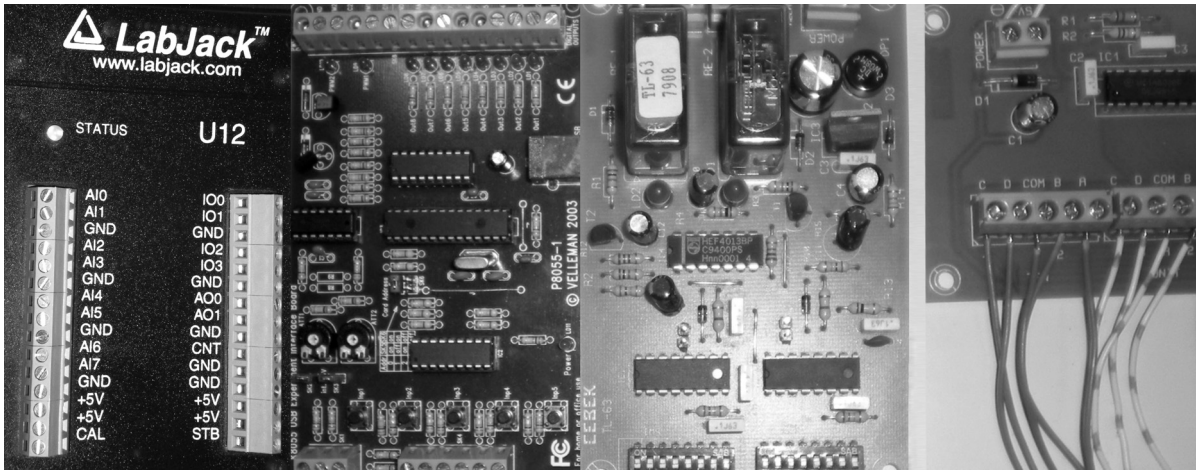


Fig. 7.9 Exemple de targetes d'entrada/sortida programables en C++

Targeta d'entrada/sortida *(I/O card)*. Són interfícies que tradueixen els senyals i les dades del dispositiu a l'estàndard que entén l'ordinador, utilitzant targetes d'entrada/sortida que actuen com a intermediàries i possibiliten la connexió de dispositius externs al port de l'ordinador.

Les targetes d'entrada/sortida (*input/output cards*) serveixen per connectar al port de l'ordinador els dispositius externs que directament no s'hi podrien connectar (v. figura 7.9).

A la figura 7.10 es pot observar un sistema de control i adquisició de dades basat en la targeta d'entrada/sortida K8055 de Velleman.

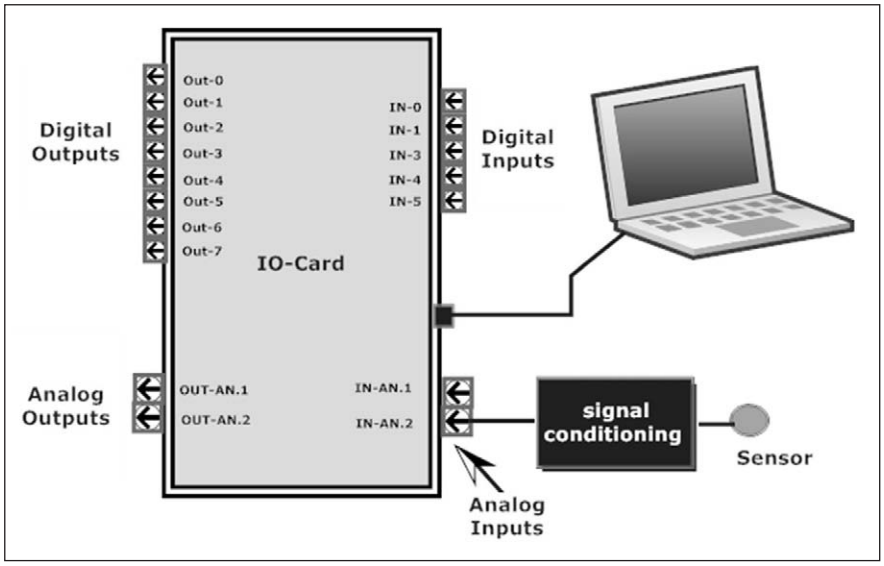


Fig. 7.10 Sistema d'adquisició de dades i control de dispositius externs basat en la *IO card*

Aquesta targeta té un conversor A/D intern que converteix el senyal procedent del condicionador a dades digitals, de tal manera que aquestes puguin ser enviades mitjançant la connexió USB (amb un cable tipus A/B que s'usa normalment per connectar la impressora) a l'ordinador. La targeta s'alimenta mitjançant la mateixa connexió USB i té 5 entrades digitals, 2 entrades analògiques, 8 sortides digitals i 2 sortides analògiques.

Per programar la targeta mitjançant C++, hem confeccionat un fitxer anomenat *IO.h* que s'ha de posar a la mateixa carpeta del programa i que s'ha d'incloure al programa mitjançant la directiva:

```
#include  ''IO.h''
```

Aquest fitxer conté la informació necessària per poder interaccionar amb la targeta, tal com es mostra a la taula 7.3.

#include "IO.h"		
Instrucció	**Valors**	**Descripció**
outport(dout,5);	Control=5 dout: 0	Establir connexió amb la targeta. Engegar la targeta
outport(dout,1);	Control=1 dout: dada per enviar	Enviar la dada dout a la sortida digital
outport(dout,2);	Control=2 dout: dada per enviar	Enviar la dada dout a la sortida analògica
din=inport(3);	Control=3 din=(B4 B3 B2 B1 B0)	Llegir l'entrada digital (5 bits: B4 B3 B2 B1 B0)
din=inport(4);	Control=4 din=entrada analògica	Llegir l'entrada analògica

Taula 7.3 Instruccions per comunicar amb la targeta IO

7.3.2 Exemple: control d'un semàfor

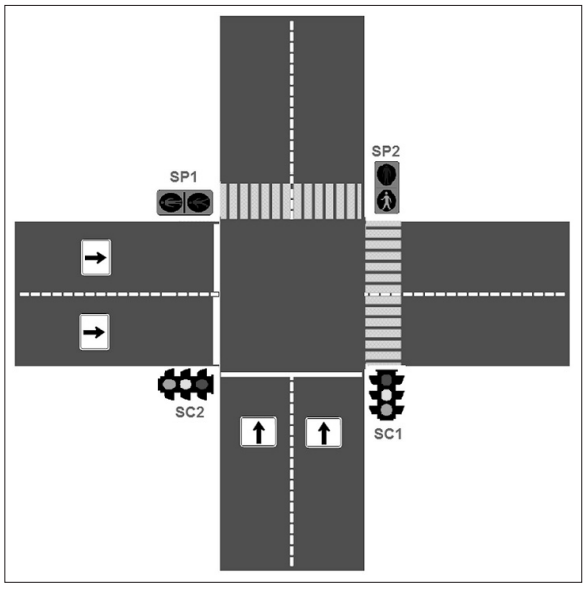


Fig. 7.11 Semàfor d'un encreuament

Es vol controlar un semàfor d'un encreuament (vegeu la figura de l'esquerra) mitjançant un sistema maquinari-programari. La seqüència del funcionament del sistema és:

- En activar ON/OFF, el sistema passa al seu estat inicial: semàfor de cotxes 1 passa a verd (SC1-G=ON) i la resta dels semàfors estaran en una combinació lògica que s'indica al diagrama d'estats, que es pot observar a la figura 7.11. En el cas d'ON/OFF=OFF, tots els semàfors s'apaguen.
- En qualsevol moment, si s'activa el polsador START (passar d'estat 0 a estat 1 generant un flanc de pujada), el sistema es reinicia i comença per la combinació inicial.
- Si en qualsevol instant s'activa l'entrada alarma (ALARM=ON), tots els semàfors passaran a vermell. Una vegada desactivat (ALARM=OFF), el sistema es reinicia i comença per la combinació inicial.

El diagrama de la figura 7.12 mostra la seqüència dels canvis en funció del tant per cent (%) del *període*= *T (green) + T (yellow) + T (red)*.

STATE	-1	0	1	2	3	4	5
SC1-R	0	1	0	0	1	1	1
SC1-Y	0	0	0	1	0	0	0
SC1-G	0	0	1	0	0	0	0
SP1-R							
SP1-G	0	0	0	0	0	1	1
SC2-R	0	1	1	1	1	0	0
SC2-Y	0	0	0	0	0	0	1
SC2-G	0	0	0	0	0	1	0
SP2-R							
SP2-G	0	0	1	1	0	0	0
Data out binary & Hex	00000000 0x00	10001000 0x88	00101001 0x29	01001001 0x49	10001000 0x88	10010010 0x92	10010100 0x94

Fig. 7.12 Diagrama d'estats del sistema del semàfor

Per al sistema de control del semàfor, es necessiten 10 sortides formades per 2 blocs de 5 bits (3 per al semàfor de cotxes i 2 per al semàfor de vianants). Com que la targeta té 8 sortides digitals, per a no usar més targetes podem considerar que és suficient controlar l'estat verd dels semàfors de vianants, ja que l'estat vermell sempre és igual que l'estat vermell del semàfor de cotxes de l'altre costat.

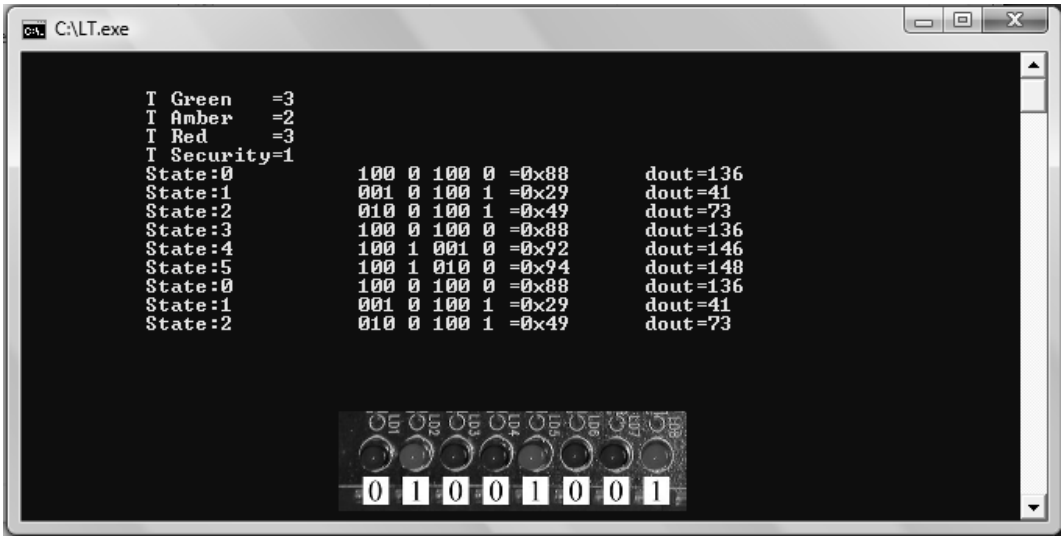
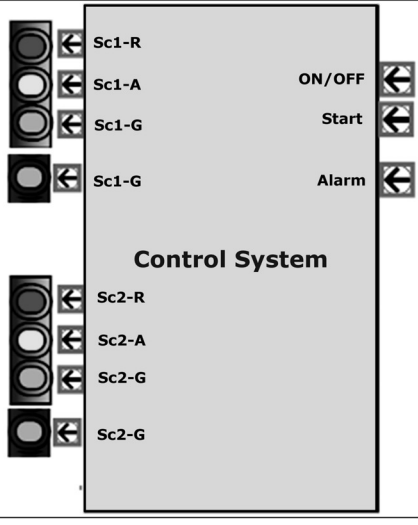


Fig. 7.13 La dada de sortida per $01001001 = 0x49$ i la visualització de dades en pantalla

Per tant, amb el programa controlem: 3 estats del semàfor de cotxes1 (SC1-R; SC1-A; SC1-G), 3 estats del semàfor de cotxes2 (SC2-R; SC2-A; SC2-G) i l'estat vermell dels semàfors de vianants (SP1-R; SP2-R). Les connexions a realitzar es mostren a la figura 7.13.



Connexió lògica de la targeta I/O	IO Card	Sistema de control
	In-0	ON_OFF
	In-1	Start
	In-2	Alarm
	Out-0	SC1-R Semàfor de Cotxe1-*red*
	Out-1	SC1-A Semàfor de Cotxe1-*amber*
	Out-2	SC1-G Semàfor de Cotxe1-*green*
	Out-3	SP1-R Semàfor de vianants1-*green*
	Out-4	SC2-R Semàfor de Cotxe2-*red*
	Out-5	SC2-A Semàfor de Cotxe2-*amber*
	Out-6	SC2-G Semàfor de Cotxe2-*green*
	Out-7	SP2-R Semàfor de vianants2-*green*

Fig. 7.14 Connexions de la targeta IO per al sistema de semàfors

Es pot observar que el sistema experimenta algun canvi respecte del seu estat anterior en sis casos. Per tant, caldria un comptador d'estats per determinar els canvis que s'han de produir.

Programa SEMAFOR.CPP

A continuació, es mostra el codi font del programa per controlar la cruïlla de semàfors amb les especificacions que s'han plantejat a la secció 7.4.2.

```
//El programa del semàfor
#include <iostream>
#include "IO.h"    // serveix per interactuar amb la targeta d'entrada/sortida. Treure // al connectar la targeta
#include <conio.h>
#include <windows.h>
// Definició de constants
#define ON 1     // ON com a estat 1
#define OFF 0    // OFF com a estat 0
#define Control_DIGITAL_OUT 1    // 1 com a control de sortida digital
#define Control_DIGITAL_IN 3     // 3 com a control d'entrada digital
#define Control_START_CARD 5     // 5 com a control d'inicialització
#define ESC 27                   // ESC= equivalent a la tecla esc en ASCII
using namespace std;
const  int Nstates=6; //Declaració de nombre d'estats màxim
const int ut=1000; //Declaració d'unitat de temps: a mes ut, més gran el cicle
// Definició tipus del vector pels temps de canvi d'estats
typedef int vector[Nstates];
// Declaració de les accions
void ini();
//ini() per definir els colors de la pantalla i el text,
//i també per connectar la targeta IO.
void end();// end() per desconnectar la targeta IO i finalitzar el programa.
void CalculateState(int& State, vector T);//calcula el següent estat del sistema
void ReadTimes(vector& T); //demana al usuari els temps del semàfor
void Process(int State); //construïu el vector d'estat per enviar a la targeta
void InPort(int& ON_OFF, int& Start, int& ALARM); //llegeix les entrades
void OutPort(int dout, int control); //envia dades a la targeta
int main()
{
  int State=0,LastState=-1, ON_OFF=OFF, Start=OFF, ALARM=OFF,KEY=1;
  vector T; ;    //vector de temps
  // crida a ini() per connectar a la targeta i posar colors a la pantalla
  ini();
  ReadTimes(T);// crida a ReadTimes() per llegir els temps del semàfor
  //bucle principal del sistema, si l'usuari no prem l'escape el sistema seguirà
  while(KEY != ESC)
   {
       InPort(ON_OFF,Start,ALARM);      // llegir l'estat de les entrades
       if (ON_OFF==OFF)State=-1;      // si l'usuari vol apagar l'estat serà -1
                                          //que vol dir apagar tots semàfors
       else{  if (ALARM==ON)State=0;    // si no, mirem si ALARM està activat,
                  else {                    //en aquest cas State=0: tots en vermell
                      if (Start==ON)State=-1;  //si no, mirem si Start està activat,
                                               //en aquest cas apagar tot    i reiniciar
                      CalculateState(State,T);  //si tot està bé, cridem a l'acció
                     }                  //CalculateState per calcular el següent estat
             }
       //si l'estat actual és diferent que l'estat anterior, procedim a canviar
       if (State != LastState)Process(State);
       LastState=State;  // renovem la variable LastState
       if (kbhit())KEY=getch();
       //si hi ha alguna tecla pulsada,ho llegim en la variable KEY
    }                                          // fi de while
```

```
}                                                    // fi de main

void CalculateState(int& State,  vector T) //acció que calcula l'estat següent
{
  static int t=0;//variable t static perquè volem que el seu valor es mantingui
  if (State==-1 || State == Nstates || t>=T[Nstates-1])t=0;
  //    si es compleixen condicions per reiniciar: la variable t= 0
  Sleep(ut); //    el sistema espera durant un temps.
  // busquem en quin estat ha d'entrar el sistema en funció del temps actual
  for (int i=0;i<Nstates;i++)
  {
    if(t<T[i]){
    //si estem en un temps inferior al temps de finalització d'un estat (i)
                State=i;          // posem State=i
                break;            // i sortim
                }
  }
  t++;        // incrementem el comptador del temps posem State=i
}

void ReadTimes(vector& T)//acció que demana a l'usuari els temps del semàfor
{
  int Tg,Ta,Tr,Ts,Tp;
  cout<<"T Green   ="; cin>>Tg;       // demanar el temps del verd
  cout<<"T Amber   ="; cin>>Ta;      // demanar el temps de l'àmbar
  cout<<"T Red     ="; cin>>Tr;        // demanar el temps del vermell
//demanar el temps de seguretat en que tots els semàfors han d'estar en vermell
  cout<<"T Security="; cin>>Ts;
  // Tg=3; Ta=2; Tr=3; Ts=1; són els valors de la taula d'estat
  // assignem a les variables de temps de cada estat el seu valor
   T[0]=Ts;        //T[0]= temps de seguretat
   T[1]=T[0]+Tg;//Temps estat 1 és el temps anterior (T[0]) més el temps de verd
   T[2]=T[1]+Ta;//Temps estat 2 és el temps anterior (T[1]) més el temps d'àmbar
   T[3]=T[2]+Ts;//Temps estat 3 és el temps anterior (T[2]) més el de seguretat
   T[4]=T[3]+Tg;//Temps estat 4 és el temps anterior (T[3]) més el temps de verd
   T[5]=T[4]+Ta;//Temps estat 5 és el temps anterior (T[4]) més el temps d'àmbar
  }

void Process(int State)
//acció que construirà el vector de sortida per enviar a la targeta
{
  int dout=0;  //Declaració i inicialització de la variable de dada de sortida
  switch(State)//segons el valor actual de State es calcula el vector de sortida
  {
       case 0:dout= 0x88;cout<<"100 0 100 0 =0x88"<<endl;
              break;//estat=3: tot vermell (seguretat)
       case 1:dout= 0x29;cout<<"001 0 100 1 =0x29"<<endl;  //estat=1: SC1 verd
              break;
       case 2:dout= 0x49;cout<<"010 0 100 1 =0x49"<<endl;  //estat=2: SC1 àmbar
              break;
       case 3:dout= 0x88;cout<<"100 0 100 0 =0x88"<<endl;
              break;//estat=3: tot vermell (seguretat)
       case 4:dout= 0x92;cout<<"100 1 001 0 =0x92"<<endl;  //estat=4: SC2    verd
```

```
                break;
        case 5:dout= 0x94;cout<<"100 1 010 0 =0x94"<<endl; //estat=5: SC2    àmbar
                break;
        default:dout=0x00;cout<<"000 0 000 0 =0x00"<<endl; // estat=-1 tot apagat
                break;
   }
 OutPort(dout, Control_DIGITAL_OUT);   // enviar    dada a la    sortida
 cout<< "state:"<<State<<": dout="<<dout<<endl;
}//fi del    Process

void OutPort(int dout, int control)
//acció que envia la dada de sortida a la targeta de IO
{
     int OK=0;// declaració de variable OK per capturar la resposta de la targeta
     //OK=outport(dout,control);
     //enviar dades a la sortida.    Treure //, en el cas de connectar la targeta
     if(OK==-1){  // si es detecta problema amb la targeta IO
                          cout<<"Error:IO-Card"<<endl;    //visualitzar l'error
                          end();    // finalitzar el programa
                          }
   }

void InPort(int& ON_OFF, int& Start, int& ALARM)//acció que llegeix les entrades
{
static int LastStart,control= Control_DIGITAL_IN,din=1;//variable estat anterior
    // din=inport(control);
    // llegir les entrades. Treure //,    en el cas de connectar la targeta
    ON_OFF=din & 1;  //fer operació AND de din amb 00000001 per verificar ON_OFF
    Start=(din & 2)>0;  //fer AND de din amb 00000010 per verificar el bit Start
    ALARM=(din & 4)>0 ;//fer AND de din amb 00000100 per verificar el bit ALARM
    if (LastStart == ON && Start == ON) Start =OFF;
    // Si ha acabat el flanc de pujada de Start, ho posem a 0
    LastStart=Start;  //operació per detectar el flanc de pujada de Start
}

void ini()
//acció per definir els colors de la pantalla, el    text,
//i també per connectar la targeta IO.
{
   int dout=0, control= Control_START_CARD;
   // preparar la dada de sortida per iniciar la targeta
   system("color 1e");
   // posem colors: el primer dígit és el color de la pantalla i el segon és
   //el color de text en hexadecimal,els colors van des de 0 (negre) a 15(blanc),
   //es a dir de 0 a f en hexadecimal. En aquet cas: blau, groc
   OutPort(dout,control);
//enviar la dada de sortida amb el control=5 que vol dir inicialitzar la targeta
}
 void end()//acció per finalitzar el programa
 {
    system("pause");
    exit(0);   //    finalitzar el programa
 }
```

7.3.3 Exemple: control d'un cotxe teledirigit

Es pot utilitzar un cotxe tipus Scalextric. S'ha de desmuntar el comandament i soldar quatre cables al circuit a cadascun dels punts de comandament i un altre cable per a la massa. D'aquesta forma, podem connectar la massa del circuit del comandament a la massa (Gnd) de la targeta i els quatre cables de direcció als primers bits de la targeta *IO*, de la manera següent:

- Enrere: [Out-3]
- Esquerra: [Out-2]
- Recte: [Out-1]
- Dreta: [Out-0]

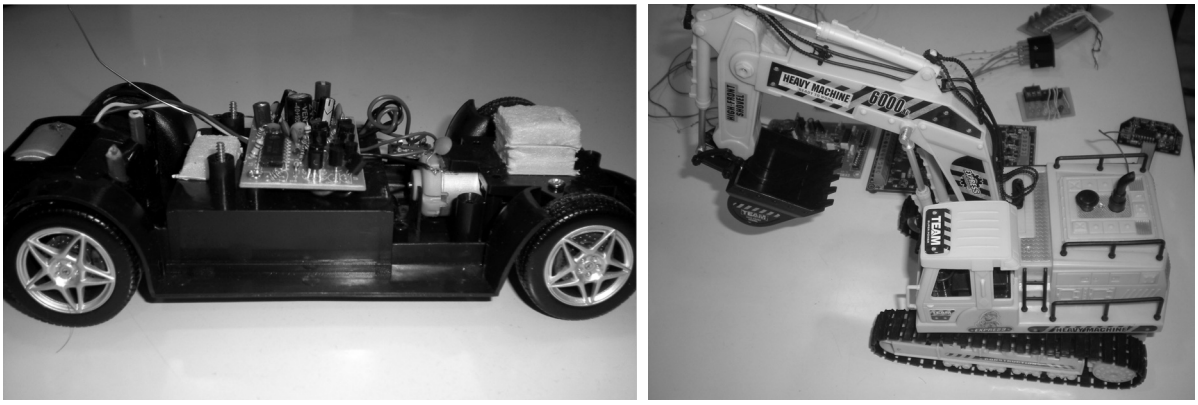


Fig. 7.15 Cotxes i grues teledirigits que pot utilitzar el programa presentat

Per programar el moviment mitjançant un programa realitzat en C++, podem tenir en compte la taula 7.4.

Enrere [Out-3]	Esquerra [Out-2]	Recte [Out-1]	Dreta [Out-0]	Dades de sortida Hexadecimal	Descripció
0	0	1	0	0x02	Recte
0	0	1	1	0x03	Girar a la dreta
0	1	1	0	0x06	Girar a l'esquerra
1	0	0	0	0x08	Marxa enrere
1	1	0	0	0x0C	Enrere-esquerra
1	0	0	1	0x09	Enrere-dreta

Taula 7.4 Dades a enviar a la targeta *IO* per diferents moviments del cotxe teledirigit

COTXE.CPP

A tall d'exemple, realitzem el programa següent per moure un cotxe en 10 voltes, automàticament.

```
int main()
{
  const int ut=100, nTURN=10;   //ut: la unitat de temps, nTURN: nº de voltes
  const int  tFORWARD=10, tTURN=2;
```

```
  // tFORWARD: nº d'unitat(ut) de temps d'anar recte, tTURN: nº d'ut per girar
  const int  dFORWARD=2,  dTURN=3;
  // dFORWAR i dTURN: les dades de sortida per anar recta per girar
  int dout=0, control=1,Lastdout=-1,nT=0,N=0,KEY=0, LASTdout=0;
  ini(); //posar color a la pantalla i inicialitzar la targeta
  while(KEY != ESC && N<4*nTURN)
  // mentre no es premi la tecla ESC i mentre nº de voltes és inferior a nTURN
   {
     Sleep(ut);     // esperar un temps equivalent a unitat de temps ut
     nT++;          // incrementar el comptador del temps
     if (nT < tFORWARD) dout=dFORWARD;
     // si el temps    és inferior al temps d'anar recta dout serà    dFORWARD
     else if (nT < tFORWARD + tTURN ) dout=dTURN;
     //sinó si és el moment de girar la dada de sortida serà dTURN
           else nT=0; N++;
     //sinó és el moment de tornar a anar recta es posa nT=0
     //i s'incrementa Nº de voltes
     if (Lastdout !=dout) OutPort(dout,control);
     //si la dout és diferent que la d'abans, s'ha de renovar la sortida
     Lastdout=dout;
     //i s'ha de refrescar el valor de dada anterior (Lastdout)
     //amb la dada actual (dout)
     cout<<":"<<nT<<":"<<dout<<endl;
     //es visualitza el temps i la dada de sortida
    }
 end();// finalitzar el programa
}
```

Nota: s'hi han d'afegir les llibreries i les accions ini(), end() i OutPort(..) del programa anterior 7.3.3.

Índex de figures

Índex de taules

Referències

J. Castro, F. Cucker, X. Messeguer, A. Rubio, Ll. Solano, B. Vallés. *Curso de Programación*. McGraw-Hill, 1992

Fatos Xhafa, Pere-Pau Vázquez, Jordi Marco, Xavier Molinero, Angela Martín. *Programación en C++ para Ingenieros*. Thomson-Paraninfo 2006.

J. Marco, F. Xhafa i PP. Vázquez. *Fonaments d'Informàtica. Pràctiques de laboratori*. Edicions UPC, 2006.

X. Franch, J. Marco, X. Molinero, J. Petit i F. Xhafa. *Fonaments de programació. Problemes resolts el C++*. Edicions UPC, 2006.

webs d'interès

http://www.cprogramming.com/ tutorials de C i C++

http://www.hello-world.com/cpp/index.php (visual C++)

http://www.bloodshed.net/dev/devcpp.html