Introducció a la programació

1

Isidre Guixà i Miranda
IES SEP Milà i Fontanals, d'Igualada

Programació estructurada i modular

Setembre del 2007
© Isidre Guixà i Miranda
IES SEP Milà i Fontanals
C/. Emili Vallès, 4
08700 - Igualada

En cas de suggeriment i/o detecció d'error podeu posar-vos en contacte via el correu electrònic `iguixa@xtec.cat`

Índex

Introducció

Aquesta primera unitat didàctica pretén introduir-vos en el món de la programació de manera que conegueu el que és un programa, el que vol dir fer un programa i els components mínims per a realitzar un programa.

Possiblement heu sentit expressions com “dissenyar programes”, “desenvolupar aplicacions informàtiques”, “llenguatges de programació”, “codificació” i moltes altres d’ús freqüent en àmbits de programació. Si no és així, potser esteu entrant en el món de la programació amb mal peu, sense tenir gaire idea de la porta que esteu travessant.

No interpreteu el paràgraf anterior, en cas que us hagueu sentit al·ludits, com una sentència desfavorable al seguiment que pugueu fer d’aquest crèdit. Simplement preteníem que us adonéssiu del que pot suposar a una persona que no ha vist mai un avió (cal tenir imaginació!) iniciar uns estudis d’enginyeria aeronàutica. Oi que pot suar una mica per tal de seguir-los amb aprofitament? En un principi li costarà, però si té les capacitats adequades i hi dedica el temps necessari, se’n sortirà i tirarà endavant. Doncs el mateix succeeix en el món de la programació.

En el nucli d’activitat "Conceptes bàsics" us pretenem situar en el món de la programació, introduint un seguit de conceptes que, a partir d’aquest moment, han de formar part del vostre llenguatge natural. En cas que els tingueu assolits us recomanem igualment el seguiment acurat del nucli, ja que és bastant comú que persones que han fet incursions en el món de la programació tinguin coneixements esbiaixats de la realitat.

En el nucli d’activitat "Dades i operadors" ens endinsarem en el món de les dades estàtiques simples que s’utilitzen en la confecció dels programes: quines dades permeten representar i quines són les operacions permeses. En primer lloc s’efectua una visió genèrica vàlida per a la majoria de llenguatges de programació i, posteriorment, es posen en pràctica en el llenguatge C.

Poca cosa podem fer si disposem de diverses possibilitats per a representar les dades, però no tenim mètodes per a gestionar-les. La programació estructurada es fonamenta en tres tractaments bàsics: seqüencial, condicional i iteratiu. En el nucli d’activitat "Instruccions seqüencials bàsiques" s’introdueix el tractament seqüencial i així es poden posar en pràctica els coneixements adquirits sobre les dades

estàtiques simples. Els dos tractaments restants (condicional i iteratiu) els estudiarem en la unitat didàctica "Estructures de control".

Per aconseguir els nostres objectius, heu de reproduir en el vostre ordinador i, a ser possible, en diverses plataformes, tots els exemples incorporats en el text, per a la qual cosa, en la secció “Recursos de contingut”, trobareu tots els arxius necessaris, a més de les activitats i els exercicis d’autoavaluació.

Objectius

A l'acabament d'aquesta unitat didàctica, l'estudiant ha de ser capaç de:

1. Valorar els programes informàtics com a eines de treball imprescindibles.

2. Reconèixer i classificar els diferents tipus de llenguatges de programació.

3. Dissenyar algorismes per a resoldre problemes quotidians.

4. Assolir les diferents fases de desenvolupament de programes.

5. Fer servir adequadament l'entorn de programació del llenguatge C: l'edició, la compilació, el muntatge i l'execució.

6. Fer servir adequadament els diferents tipus de dades estàtiques simples per a les dades constants i variables en el disseny d'algorismes en pseudocodi.

7. Dissenyar algorismes seqüencials en pseudocodi per a resoldre problemes que permetin una solució seqüencial.

8. Identificar i aplicar la representació gràfica per l'estructura seqüencial.

9. Aplicar l'estructura seqüencial i els tipus de dades simples en la codificació de programes en llenguatge C.

10. Fer servir de manera adequada el depurador del llenguatge C per al seguiment de l'execució dels programes.

1. Conceptes bàsics.

En iniciar qualsevol aprenentatge acostuma a ser necessari la introducció d'un seguit de conceptes bàsics i aquest fet es manté en el camp de la programació. Així, ens proposem introduir conceptes com programació, programa, algorisme, fases de desenvolupament, tipus de programació. llenguatges de programació i una primera incursió en el llenguatge C

1.1. Programació i programa

Què és un ordinador? El podem definir com una màquina que ens permet aconseguir el tractament de dades d'una manera automàtica. Però l'ordinador no és intel·ligent i, per tant, no executarà el tractament esperat si ningú no l'ha preparat per a aquesta tasca. Aquesta preparació s'anomena programació i té com a resultat final el programa que, quan és executat per l'ordinador, permet aconseguir l'automatisme que es pretenia.

Un programa és una seqüència d'ordres, comprensibles per l'ordinador, que permeten la realització d'una tasca determinada.

La programació és la seqüència de passos que ha d'efectuar el programador per a obtenir un programa.

Intel·ligència artificial

En l'actualitat els ordinadors no tenen capacitat de pensar. Potser heu sentit parlar de la intel·ligència artificial. No caigueu en paranys. Els ordinadors són màquines creades per l'ésser humà i només poden fer les feines per a les quals han estat programats.

Exemple de programació

Considerem el típic exemple que es proposa a tot estudiant de programació en la seva primera classe.

Volem ensenyar a algú (persona o màquina) com ha de fer una truita francesa de dos ous. Som-hi!

Tots tenim clar el que cal fer. Els passos són els següents.

a) Cal preparar un plat, una forquilla, una paella, l'oli, la sal, els dos ous i tenir un fogó.
b) Es trenquen els dos ous dins el plat sense que hi caigui cap tros de closca.
c) Es posa una mica de sal sobre els rovells.
d) Es bat la clara i els rovells amb la forquilla.
e) Cal posar un rajolí d'oli a la paella i escalfar-la en el fogó.
f) Quan l'oli estigui calent, s'aboca el contingut del plat a la paella.
g) A mesura que el contingut va quallant, es plega sobre si mateixa amb l'ajut de la forquilla.
h) Cal treure la paella del foc quan el contingut hagi quallat del tot.

Bé, ja tenim la truita feta. Bon profit!

A l'exemple anterior hem elaborat un programa que permet aconseguir un objectiu final: fer una truita francesa de dos ous. Aquí tenim el resultat final, el programa, però per arribar-hi hem hagut de seguir, en l'ordre següent,

1) una fase d'anàlisi, pensant detalladament tot el que suposa fer una truita;

2) una fase de disseny, lligant de manera adequada totes les idees aparegudes a la fase d'anàlisi;

3) una fase d'escriptura, redactant de manera ordenada i clara la seqüència de passos que cal seguir.

Els passos anteriors constitueixen, a grans trets, el procés de programació i, el resultat final, la redacció dels passos que van des de la a) fins a la h), és el programa en llengua catalana, de manera que qualsevol persona amb suficient coneixement d'aquesta llengua el podria seguir per aconseguir fer una truita francesa de dos ous. Ara bé, el resultat final, tal com el tenim, no serveix per a una persona sense coneixements de llengua catalana.

D'aquest exemple cal que poseu especial atenció en uns termes importants que tractarem en apartats posteriors: "fase d'anàlisi", "fase de disseny", "fase d'escriptura" i "llengua".

Per acabar, també hauria de quedar clar quan té sentit desenvolupar un programa, atès que fer-ho no és simple. Bàsicament, és pertinent en dos casos:

1) Quan l'execució del programa s'ha de fer un gran nombre de vegades, és a dir, quan el programa permet automatitzar el desenvolupament de tasques, complexes o no, de caràcter repetitiu, com és el cas d'una cadena de muntatge, la confecció de nòmines dels treballadors, etc.

2) Quan el programa serveix per a assegurar l'execució, unes poques vegades, d'una tasca complexa i feixuga, en la qual s'ha d'assegurar fiabilitat i rapidesa, com és el cas del control d'una nau espacial.

És a dir, no té cap sentit desenvolupar un programa per a resoldre un problema que ens trobarem poques vegades o que és de molt fàcil resolució. En tal cas, té més sentit fer el tractament manual que calgui que no pas realitzar el programa corresponent. ‼

1.2. Algorismes

Siguem ara més rigorosos i comencem a conèixer i a utilitzar la terminologia correcta en l'àmbit de la programació.

Una aplicació informàtica, que no és altra cosa que un conjunt de programes enllaçats de manera convenient, té per objectiu fer més àgils els processos i disminuir el nombre de fases de desenvolupament que s'hi esmercen. Per això, la realització o el disseny de programes comporta el seguiment metòdic d'un procés d'execució.

La solució del problema que planteja l'usuari ha de passar necessàriament per la realització d'una seqüència d'accions determinades. Aquesta seqüència constitueix un algorisme.

Usuari

Usuari és el terme que s'utilitza per a referir-se a la persona que fa servir una aplicació informàtica amb la intenció d'aconseguir un resultat final.

> Un algorisme és un mètode de resolució d'un problema en un nombre finit de passos.

Algorisme no significa la resolució d'un problema particular per una situació particular, sinó la resolució de tots els problemes del mateix tipus, sigui quina sigui la situació de partida i preveient les alternatives d'actuació convenients segons les diferents situacions que es puguin presentar. (!!)

Exemple d'algorisme

Tornem a l'exemple de la preparació d'una truita francesa de dos ous.

Aquell programa es basa en un algorisme o seqüència finita de passos que s'han de seguir, el qual és utilitzable a qualsevol lloc. És a dir, l'algorisme no depèn de la cuina on s'hagi de fer la truita, ni de com són els ous (rossos o blancs), ni de quina paella fem servir, ni de com és la forquilla. L'algorisme ens serveix per a fer una truita mentre tinguem tots els ingredients.

Ara bé, continuant amb el mateix algorisme, cal remarcar que no preveu certes situacions que es poden donar, com és el cas del que cal fer si manca algun dels ingredients de l'apartat a).

Sembla clar que hi ha ingredients que no hi poden faltar: els ous i el fogó. Però els altres potser són substituïbles o eliminables. Expliquem-nos: una paella es podria substituir, per exemple, per un altre tipus d'utensili. La forquilla segur que és substituïble. La sal ja és més difícil, però el programa davant la detecció de la manca de sal potser podria preguntar al futur comensal si li va bé sense sal i en cas afirmatiu continuar amb la preparació de la truita o avortar el procés en cas contrari.

Segur que els professionals hostalers ens en podrien impartir lliçons i lliçons, de com fer truites franceses.

> Anomenem processador tota entitat capaç d'executar un algorisme donat.

En l'exemple de la truita francesa, tota persona que sàpiga llegir català i que disposi de les eines (o els ingredients) necessaris pot ser un processador adequat per a aconseguir la truita.

El conjunt d'eines necessàries per a executar un algorisme constitueix l'entorn de l'algorisme.

Entorn i algorisme estan estretament lligats, ja que la majoria de vegades el disseny de l'algorisme depèn de l'entorn, és a dir, de les eines posades a disposició del processador.

Màquina expenedora

Les màquines expenedores són un bon exemple de l'ús d'algorismes en la vida real.

Entorn i algorisme: el cas d'un expenedor de begudes

Imaginem una màquina de begudes que rep, per part del comprador, un conjunt de monedes i una sol·licitud de compra (la beguda desitjada).

L'entorn de l'algorisme que s'ha d'executar està format pels actes executats pel comprador (el conjunt de monedes introduïdes i la sol·licitud de beguda) i per l'estoc de begudes i monedes que hi ha a la màquina (per tal de poder servir la beguda demanada i de retornar el canvi que correspongui).

És prou clar que les decisions que ha d'executar la màquina depenen de l'entorn. Per tant, l'algorisme dissenyat ha de tenir en compte totes les situacions possibles. El seu disseny ha estat en funció de l'entorn.

Per què no comenceu a pensar com es dissenya l'algorisme que executa la màquina de begudes?

Per tant, un algorisme és la descripció exacta i sense ambigüitat de la seqüència de passos elementals que cal aplicar per, a partir de l'entorn del problema, trobar la solució cercada. Perquè un algorisme sigui complet haurà de preveure totes les alternatives lògiques possibles que les diferents combinacions d'entorns possibles puguin presentar. Cadascun dels passos d'un algorisme s'anomena sentència o instrucció.

Una instrucció és una combinació de paraules, dades i símbols que, obeint la sintaxi pròpia del llenguatge, són utilitzats per l'ordinador per a dur a terme una acció determinada.

Si ens centrem en el cas que el processador sigui un ordinador, l'entorn del problema està constituït per un conjunt d'informacions que ha de fer servir l'ordinador i que anomenem dades.

Anomenem dada tota informació que fa servir l'ordinador.

Amb els conceptes de dada i instrucció introduïts ja podem dir que tot algorisme ha de complir dues condicions fonamentals:

1) descripció, mitjançant instruccions, de les accions (passos) que s'han d'executar;

2) descripció, mitjançant declaracions i definicions, de les dades que són manipulades per aquestes instruccions.

> Un programa és l'expressió d'un algorisme en un llenguatge entenedor per a l'ordinador.

No hi ha un llenguatge únic i universal per a descriure els algorismes. En aquest crèdit treballarem amb un llenguatge teòric (pseudocodi) i un llenguatge real (C).

1.2.1. Classificació d'algorismes segons el flux d'execució

Segons el flux d'execució, els algorismes es poden classificar en:

1) Lineals o seqüencials, en què les instruccions s'executen en el mateix ordre en què s'han escrit.

> Tornant a l'exemple de la truita francesa de dos ous, l'execució ordenada de tots els passos fa que l'algorisme sigui lineal.

2) Cíclics o iteratius, en què un grup de línies s'executa un nombre finit de vegades.

> A l'exemple de la truita francesa de dos ous, hi ha un apartat que consisteix a esperar que l'oli sigui calent. En realitat, aquesta espera no és cap altra cosa que l'execució continuada d'un cicle semblant a esperar cinc segons mentre l'oli no sigui calent.
>
> És a dir, esperem cinc segons i mirem si l'oli és calent. Si no és així tornem a esperar cinc segons, i així successivament fins que arriba un moment que podem assegurar que l'oli és calent. Si no ho poguéssim assegurar, el programa mai no sortiria d'aquest cicle i s'hauria quedat en un cicle infinit.

3) Condicionals o alternatius, en què hi ha certes condicions que provoquen l'execució de fases diferents del programa depenent del fet que es compleixin o no aquestes condicions.

> A l'exemple de la truita francesa de dos ous, hem fet notar, posteriorment, que l'algorisme s'hauria pogut millorar de manera que s'hagués pres decisions en funció dels ingredients existents (recordeu què es podia fer si hi mancava la sal, la forquilla, la paella, etc.)
>
> En aquests casos estaríem davant una situació condicional, en què l'algorisme variaria la seva execució en funció de la situació detectada o de les respostes efectuades per l'usuari.

En realitat és molt difícil trobar un algorisme que es classifiqui exactament en un dels tres casos anteriors. El que acostuma a passar és que un algorisme és una seqüència lineal d'accions, algunes de les quals són iteratives i d'altres, condicionals. (!!)

1.2.2. Característiques desitjables per als algorismes

Quan es dissenya un algorisme s'ha de procurar que tingui les característiques següents:

- finit, és a dir, cal assegurar que finalitza;
- llegible, o sia, que estigui escrit de manera que sigui fàcil de llegir i entendre;
- portable, de manera que pugui ser instal·lat en diferents tipus d'ordinador;
- modificable, ja que així les modificacions i actualitzacions necessàries per a una nova situació del programa seran fàcils de fer;
- eficient, perquè ocupi tan poc espai com sigui possible i així s'aprofiti al màxim la memòria de l'ordinador i s'aconsegueixi que el temps d'execució sigui mínim;
- estructurat, situació que s'aconsegueix utilitzant de manera adequada les regles de la programació estructurada;
- modular, de manera que el programa, anomenat programa principal, pugui estar subdividit en mòduls o programes més petits, si tenen raó de ser per si mateixos, anomenats subprogrames, cadascun dels quals realitza una part del programa principal.

1.2.3. Representació d'algorismes

Hi ha diferents representacions utilitzades per a confeccionar algorismes. Coneguem-les:

1) Diagrames de flux o ordinogrames, basats en símbols gràfics per a la seva resolució.

2) Taules de decisió, que permeten tabular totes les possibles situacions que es poden presentar en el problema i les corresponents accions que cal prendre per a cadascuna d'aquestes.

3) Pseudocodi, que permet descriure un algorisme utilitzant una barreja de frases en llenguatge comú, instruccions de llenguatge de programació i paraules clau que en defineixen les estructures bàsiques.

El mètode que s'utilitza en programació estructurada, objectiu final d'aquest crèdit, és el pseudocodi. Els altres, en l'actualitat, estan fora d'ús pel que fa al disseny genèric d'algorismes. No obstant això, en algunes ocasions es fan servir per a acompanyar l'algorisme en pseudocodi i ajudar a la seva comprensió.

1.3. Cicle de vida d'una aplicació informàtica

El desenvolupament d'una aplicació informàtica comprèn les fases que esmentem a continuació i que veurem en detall en els subapartats següents.

1) Anàlisi prèvia. Definició del problema.

2) Anàlisi funcional.

3) Anàlisi orgànica.

4) Programació de l'aplicació informàtica, amb les subfases codificació, traducció, muntatge i verificació

5) Posta en marxa (instal·lació i explotació).

Una vegada posada en marxa l'aplicació (i en ocasions, fins i tot, abans) pot donar-se el cas que la problemàtica inicial per la que es va desenvolupar l'aplicació hagi canviat i, per tant, calgui efectuar retocs a l'aplicació informàtica per tal d'adaptar-la als nous requeriments.

Estem, en aquest cas, davant la problemàtica d'efectuar una adaptació a una aplicació informàtica (finalitzada o no) i per a fer-ho, cal tornar a aplicar les cinc fases sobre el material ja existent (no es tracta de partir de zero!). Apareix, per tant, un cicle que fa que l'aplicació informàtica es trobi en continua evolució. S'acostuma a anomenar cicle de vida de l'aplicació informàtica. ‼

1.3.1. Anàlisi prèvia. Definició del problema

La definició del problema és el primer pas del procés que cal seguir per a resoldre'l. Cal tenir molt clar què és el que s'ha de resoldre. I això implica una molt bona comunicació amb l'organització que encarrega la

realització del programa i amb els usuaris finals, establint amb precisió i claredat els objectius que es volen assolir.

1.3.2. Anàlisi funcional

Una vegada el problema està definit, cal estudiar-ne la resolució, cercant les diferents opcions i escollint-ne la millor, la qual caldrà valorar en termes d'eficiència i de costos de realització.

En aquesta fase s'acostuma a:

- Parlar amb tots els usuaris implicats (des del director o gerent fins al conserge, si és el cas) de manera que aconseguim tenir tots els punts de vista davant el problema.

- Donar totes les solucions possibles amb un estudi de viabilitat, considerant els costos econòmics (material i personal).

- Proposar al client, futur usuari de l'aplicació, una solució d'entre les possibles, i ajudar-lo a prendre la decisió més adequada.

- Trencar la solució adoptada en parts independents, anomenades unitats funcionals, per a facilitar el tractament per separat en la fase següent.

Totes les fases de desenvolupament d'un programa són importants, però aquesta és, potser, la fonamental, ja que és la fase en què es prenen les decisions. ‼

> Els analistes funcionals són les persones que desenvolupen aquesta fase d'anàlisi funcional.

L'anàlisi no és responsabilitat del programador. Això no implica, però, que no hagi de ser-ne coneixedor. Al contrari, doncs el programador haurà de desenvolupar programes a partir d'anàlisis efectuades per analistes.

Bé, futurs programadors, no entrarem en les fases d'anàlisi, però, al llarg del crèdit, per cada problema que calgui resoldre, haureu de fer-ne l'estudi corresponent per tal de dissenyar una solució, intentant que sigui, a més a més, la millor solució.

1.3.3. Anàlisi orgànica. Disseny dels algorismes

En aquesta fase, destinada als analistes orgànics, es dissenya l'algorisme de tractament corresponent a cada unitat funcional definida en la fase anterior.

Ja hem dit més amunt que dissenyaríem els algorismes en pseudocodi i, puntualment, amb taules de decisió i amb ordinogrames.

El pseudocodi és un llenguatge que fem servir per a escriure l'algorisme d'una manera entenedora, utilitzant paraules del nostre llenguatge. Ara bé, hi ha algunes paraules clau d'obligada utilització i que es consideren reservades, és a dir, n'hi ha que no es poden fer servir per a cap altra cosa que per a l'ús que tenen assignat.

Paraules reservades

Les paraules reservades no són una característica exclusiva del pseudocodi. Tot llenguatge de programació també en té.

En la realització d'un algorisme en pseudocodi se segueixen les regles que detallem: ‼

1) Les paraules clau s'escriuen diferenciant-les de la resta de paraules d'alguna manera com, per exemple, subratllant-les. En el cas de material imprès, s'acostuma a fer amb negreta o amb el tipus de lletra courier. En aquest material farem servir la lletra negreta. L'informàtic, quan fa els seus algorismes en paper, acostuma a utilitzar el subratllat. Per tant, ja sabeu què us toca: heu de subratllar les paraules reservades del llenguatge en qüestió.

2) En els algorismes s'utilitza el concepte de bloc per a agrupar un conjunt d'instruccions del "mateix nivell". Doncs bé, el text interior d'un bloc, l'escriurem sagnat respecte del marge de la capçalera. D'aquesta manera es poden distingir visualment els diferents blocs que conté el programa.

3) Tota instrucció s'ha d'acabar amb un punt i coma (;).

Un programa en pseudocodi ha de tenir, com a mínim, l'estructura següent:

```
programa <nom> és
   <cos del programa principal>
fiprograma
```

Les expressions emmarcades entre els símbols "<" i ">" indiquen que corresponen a paraules que pot decidir el programador, sense fer servir cap de les paraules reservades.

Bé, fem el típic programa que s'acostuma a dissenyar en l'aprenentatge de tots els llenguatges i que consisteix en aconseguir que l'ordinador ens doni el missatge "Hola, món!".

Per a aconseguir això, necessitem una manera d'obligar l'ordinador que faci el que nosaltres volem, que no és altra cosa que donar un missatge. És a dir, volem que l'ordinador s'expressi, ja sigui parlant (per l'altaveu o per mòdem) o escrivint (per la pantalla o la impressora). Qualsevol d'aquestes formes d'expressió s'aconsegueix mitjançant el que en programació s'anomena instrucció de sortida.

A partir d'ara disposarem en pseudocodi de la instrucció de sortida següent:

```
escriure ("<text del missatge>");
```

que suposarem que treu el missatge per la pantalla. De moment no ens ha de preocupar el fet d'utilitzar altres canals de sortida com ara la impressora, l'altaveu, etc.

Per tant, ja podem escriure l'algorisme:

```
programa primer_programa és
   escriure ("Hola, món!");
fiprograma
```

Enhorabona! Ja heu escrit el vostre primer algorisme en pseudocodi.

1.3.4. Programació de l'aplicació informàtica

Una vegada tenim definit l'algorisme (o sigui, escrit en pseudocodi), cal transmetre'l a la màquina. I això ho farem amb un conjunt de passos (codificació, traducció, muntatge i verificació) que s'han de seguir de manera ordenada.

Cadascun dels passos següents ha d'estar ben documentat, per tal que qualsevol informàtic, fins i tot el mateix programador quan ja ha passat un cert temps, pugui retocar l'aplicació si és necessari. Penseu que en un programa informàtic s'automatitza una forma d'actuació, la qual cosa implica reflectir un cert coneixement. Però el coneixement no és únic i, per tant, entendre el que un altre informàtic o conjunt d'informàtics va decidir en una situació pretèrita és molt difícil. La documentació és l'única manera de garantir aquest coneixement.

L'informàtic

Un informàtic és un artista que aconsegueix que una màquina tingui un determinat comportament, però, a diferència de l'artista típic, no es pot permetre la llicència de no documentar-ho. Ha de deixar la seva obra de tal manera que un altre artista pugui continuar-la, reformar-la, retocar-la, etc.

Codificació. Llenguatges de baix i alt nivell

La fase de codificació consisteix a reescriure l'algorisme en un llenguatge especial que puguem fer entendre a l'ordinador. Per a aquest objectiu tenim els llenguatges de programació.

Els llenguatges de programació es poden classificar en dos grans apartats: llenguatges de baix nivell i llenguatges d'alt nivell.

> Un llenguatge de baix nivell és un llenguatge que pot ser entès directament per l'ordinador.

Per a nosaltres, el català és un llenguatge de baix nivell, ja que l'entenem directament i no necessitem cap traducció. En canvi, per a la immensa majoria de nosaltres, el xinès no és un llenguatge de baix nivell, ja que necessitem algú que ens el tradueixi si volem entendre el missatge que vehicula.

Els ordinadors, conjunt de ferralla i components electrònics, només entenen impulsos elèctrics (hi ha corrent o no n'hi ha). Dit en altres paraules, l'ordinador sap treballar amb dos estats, cosa que s'ha convingut a representar amb zeros i uns.

> El llenguatge màquina és un llenguatge consistent en combinacions de 0 i 1 agrupades en octets.

Escriure un programa en llenguatge màquina és feina de… Els errors són a l'ordre del dia. Per aquest motiu s'han incorporat al món de la programació uns altres llenguatges de baix nivell.

> El llenguatge assemblador és un llenguatge constituït per un conjunt de símbols mnemotècnics que representen cadascuna de les operacions que el processador és capaç d'executar.

Un programa escrit en llenguatge assemblador es tradueix a llenguatge màquina amb eines subministrades pels mateixos fabricants del maquinari.

Treballar amb els llenguatges de baix nivell implica conèixer l'arquitectura del processador en què es basa l'ordinador i, com que de

processadors n'hi ha de diferents característiques, codificaríem un programa que només serviria per a ordinadors amb processador igual o equivalent. Aquest fet provoca que no es programi en llenguatges de baix nivell.

Els llenguatges d'alt nivell són llenguatges propers al llenguatge humà i independents del tipus de processador.

Això ja ens agrada més. Podrem escriure programes en "llenguatge habitual" per a nosaltres. Sabeu anglès, oi? Doncs us agradarà saber que els llenguatges d'alt nivell acostumen a ser propers a l'anglès.

Anglès i informàtica

Dominar l'anglès tècnic és imprescindible per a un informàtic, ja que és l'idioma vehicular principal de la programació.

Ah! Un comentari: per a escriure el programa caldrà utilitzar un editor de textos. La majoria de llenguatges d'alt nivell ja en porten incorporat un en el seu entorn de treball. Però si no us agrada podeu fer servir el que millor us sembli. Només heu de tenir una precaució: enregistrar-lo en format textual.

Traducció. Entorns intèrprets i entorns compilats

Els programes escrits en llenguatges d'alt nivell són fàcils de llegir, modificar i transportar entre diferents plataformes. Ara bé, aquestes característiques són possibles perquè hi ha unes eines que tradueixen el programa escrit en llenguatge d'alt nivell al llenguatge màquina de la plataforma que correspongui.

El programa font és el resultat de la codificació d'un algorisme en un llenguatge d'alt nivell.

A partir del programa font hi ha dues possibilitats de treball, en funció de l'entorn de programació. (!!)

Un entorn de programació interpretat tradueix el codi font a codi màquina i l'executa, instrucció per instrucció.

Un entorn de programació compilat tradueix tot el codi font a codi màquina i obté un nou programa anomenat programa objecte.

Qüestions de vocabulari

- A vegades es parla del codi font en lloc de programa font.
- Es pot anomenar codi objecte el programa objecte.
- Podem parlar de programes compiladors o simplement compiladors per referir-nos a programes traductors.

El procés de compilació, dut a terme per un programa traductor, avalua la sintaxi emprada en tot el codi font, i, en cas d'errors, dóna els corresponents avisos, sense arribar a generar el codi objecte. En tal situació, el programador ha de revisar el codi font i repetir el procés fins a aconseguir el codi objecte.
L'avaluació de la sintaxi emprada en un procés interpretat s'efectua instrucció per instrucció durant l'execució del programa. Per tant, i atès que un programa pot tenir fluxos de caire alternatiu, pot succeir que un error de sintaxi no aparegui fins que no es produeixi una determinada situació que provoqui l'execució d'una part no executada mai abans. Això és un inconvenient de l'entorn interpretat davant l'entorn compilat. Un altre inconvenient és la pèrdua de temps que comporta traduir el programa a cada execució.

Cal distingir els errors de compilació dels errors d'execució. Quan obtenim el codi objecte ens assegurem que no hi hagi errors de compilació. Hi pot haver, però, errors detectats en temps d'execució, que són produïts per situacions no controlades pel programador, és a dir, per errors en el disseny de l'algorisme.

Generació d'executables. Execució amb *runtime*

Els entorns compilats s'agrupen en dues categories en funció de la forma d'execució: generació de codi executable o execució amb *runtime*.

Un programa muntador és un programa que genera un programa executable a partir del(s) programa(es) objecte(s).

Qüestions de vocabulari

A vegades es parla de codi executable en lloc de programa executable.

El procés de muntatge avalua la coherència dels diferents codis objectes muntats juntament amb les llibreries necessàries per a l'execució del programa. En cas d'errors, dóna els corresponents avisos, sense arribar a generar el codi executable. En tal situació, el programador ha de revisar el(s) codi(s) font i repetir el procés fins a aconseguir el codi executable. Un programa executable estaria format per diversos programes objectes en cas que s'hagués dissenyat la solució al problema subdividida en diferents apartats (unitats funcionals o equivalents).

Linker

El terme anglès utilitzat per a referir-se al programa muntador és el de *linker*. Aquest terme es fa servir col·loquialment en expressions com "ara *lincaré* l'aplicació", "error de *lincatge*", etc; però el terme català que correspon a *link* és "enllaç", i el verb és "fer un enllaç".

Podem imaginar-nos una llibreria com un conjunt d'eines que els programes necessiten per a la seva execució i que depenen del llenguatge de programació emprat. Per això, en la fase de muntatge s'ajunten codi(s) objecte i llibreries.

Entenem per programa executable un arxiu (en llenguatge màquina) que es pot executar des del sistema operatiu en qüestió. Per exemple, un programa executable en el sistema operatiu DOS el distingim perquè té extensió .exe o .com. En altres sistemes, com per exemple UNIX (Linux), no és necessari que tinguin cap extensió especial.

Hi ha, però, alguns entorns compilats que no tenen la possibilitat de generar un programa executable. En tal cas parlarem d'execució amb *runtime*.

> Una execució amb *runtime* consisteix a executar un programa objecte per mitjà d'un programa que és el conductor de l'execució, el qual s'anomena *runtime*.

Ja hem comentat que per a l'entorn intèrpret, no hi ha fase de compilació que generi codi objecte i, per tant, l'execució parteix del propi codi font. En aquest cas també hi ha un *runtime* que condueix la traducció i l'execució.

L'execució d'un compilat amb *runtime* és molt més ràpida que l'execució d'un interpretat (que també necessita un *runtime*), ja que la traducció ja està feta. A més, en el compilat ja hem assegurat una sintaxi correcta.

Verificació. Depuració

Una vegada tenim el programa per a ser executat (sigui quin sigui l'entorn) cal comprovar que el programa actua tal com s'havia decidit que havia d'actuar.

> Un joc de proves és un conjunt de situacions que permet provar la bona actuació del programa. Aquest conjunt ha de ser complet en el sentit que ha d'abastar totes les possibilitats reals.

És a dir, primer caldrà dissenyar i preparar el joc de proves per, posteriorment, executar el programa per a les diferents situacions preparades, de manera que se'n pugui comprovar el bon funcionament en tot moment.

En cas que sempre vagi bé, perfecte! Podem estar més o menys segurs de la fiabilitat del programa. La realitat és que sempre s'escapen

situacions no previstes (això vol dir una anàlisi defectuosa del problema). Aquestes es detectaran, quin remei!, en la fase d'explotació.

Què hem de fer quan veiem que no funciona? Cal cercar on està el problema, que pot residir en:

a) disseny del(s) algorisme(s),
b) codificació incorrecta (però amb sintaxi correcta, ja que ha compilat).

A vegades l'error es troba de manera senzilla amb un simple seguiment de l'algorisme dissenyat. Però a vegades, la majoria de les vegades, l'error costa de trobar perquè ens entossudim que l'algorisme és correcte. En tal cas no hi ha cap altre remei que començar la depuració del programa.

Un programa depurador és un programa que permet:

Programa depurador

El terme anglès equivalent a depurador és debugger.

- Executar un programa instrucció per instrucció i veure què passa després de l'execució de cada una.
- Obtenir el valor de les dades abans i després d'executar una instrucció.
- Modificar el valor de les dades durant l'execució.
- Interrompre o aturar l'execució del programa en qualsevol punt.

La majoria d'entorns de programació actuals disposen d'eines de depuració. Si no és el cas, el programador no tindrà més remei que activar manualment la depuració, mètode que consisteix a intercalar instruccions de seguiment (missatges per on passa el programa, valors de les dades, etc.) enmig del codi del programa, per tal de fer el seguiment de l'execució i localitzar la causa del problema.

1.3.5. Posta en marxa (instal·lació i explotació)

Una vegada es té el programa final (executable o objecte o font, tot dependrà de l'entorn), cal instal·lar-lo seguint les instruccions de l'entorn i les directrius de l'administrador del sistema informàtic i, una vegada instal·lat, posar-lo en funcionament (fase d'explotació).

Administrador del sistema informàtic

L'administrador del sistema informàtic és la persona que té la responsabilitat, entre d'altres, d'assignar espai a les diferents aplicacions que s'han d'executar.

Sembla que arribem al final del procés però encara ens queda una acció: formar l'usuari. Cal tenir paciència. Pensem que nosaltres hem estat molt de temps pensant en el problema i treballant en la seva solució. Per a nosaltres és la solució normal, la tenim assumida, però per a l'usuari no té per què ser la solució desitjada o esperada. Per tant caldrà tenir una bona dosi de paciència per a explicar el funcionament de l'aplicació.

La formació de l'usuari ha d'anar acompanyada, sempre, de la guia o manual de l'usuari. No ha de ser un llibre de difícil lectura; haurien de ser els apunts que li permetessin moure's ràpidament per l'aplicació, tractant totes les possibilitats d'una manera ràpida i clara.

1.4. Tipus de programació.

Hi ha diferents tipus de programació depenent dels mètodes i de les tècniques emprades. Així podem parlar de programació estructurada, programació modular, programació orientada a objectes, programació concurrent, programació funcional, programació lògica, etcètera. Recordem que l'objectiu d'aquest crèdit, com el seu nom indica, és la programació estructurada i modular.

Cadascun d'aquests tipus de programació ha comportat l'aparició de llenguatges de programació d'alt nivell específics.

1.4.1. Programació estructurada

La programació estructurada va néixer a finals dels seixanta (1960) com a solució als problemes que presentava la tècnica de programació utilitzada fins el moment, la qual no tenia cap nom explícit, però que es va anomenar programació no estructurada a partir de l'aparició de la programació estructurada.

Un programa no estructurat és un programa procedimental, en el que les instruccions s'executen en el mateix ordre en que han estat escrites i utilitza la instrucció `goto` (anar) per passar el control a qualsevol altra part del programa. En executar-se una instrucció `goto`, la seqüència d'execució del programa continua a partir de la instrucció indicada pel `goto`. D'aquesta manera, per entendre com funciona un programa és necessari simular la seva execució i això vol dir que en la majoria dels casos és fa molt difícil comprendre la lògica del programa.

Observem la següent seqüència d'instruccions corresponents a un possible algorisme no estructurat per a un robot dedicat a rentar els utensilis emprats en una cuina, amb la responsabilitat de deixar els plats en una pila de plats i la resta d'utensilis rentats en una pila genèrica:

```
1. agafar_objecte;
2. posar_sabó;
3. passar_el_fregall_per_l'objecte;
4. si_queda_brutícia_goto_a_la_instrucció_2;
5. si_l'objecte_és_un_plat_goto_a_la instrucció_9;
6. posar_objecte_a_la_pila_genèrica;
```

```
7. si_queden_objectes_per_netejar_goto_a_la_instrucció_1;
8. goto_a_la_instrucció_11;
9. posar_objecte_a_la_pila_de_plats;
10. goto_a_la_instrucció_7;
11. netejar_la_pica;
12. informar_de_final_de_procés;
```

Oi que es fa una mica difícil fer el seguiment d'aquestes 12 instruccions? I això que coneixem l'objectiu final de les 12 instruccions! Ara imagineu-vos que arriba a les nostres mans un codi similar però amb mils de línies i hem de fer-ne el seguiment per tal d'efectuar-hi alguna modificació o cercar-hi un error... Vaja, que potser és millor llençar-ho i tornar-ho a fer partint de zero!

La programació estructurada utilitza únicament tres tipus d'instruccions (seqüència, instrucció condicional i instrucció repetitiva) amb les quals es pot desenvolupar qualsevol programa informàtic i no permet la utilització d'instruccions de transferència incondicional com `goto`. (!!)

1) La seqüència indica que les instruccions es llegiran de principi a fi.

2) La condicional indica que segons una condició s'executarà un conjunt d'instruccions o un altre.

3) La repetitiva indica que un conjunt d'instruccions, sota una certa condició, es podria repetir un determinat número de vegades.

A més, es recomana utilitzar el sagnat en les instruccions imbricades en altres per a facilitar el seguiment i la comprensió. (!!)

Observem la versió estructurada de l'algorisme per al robot que renta els utensilis emprats en una cuina:

```
mentre hi_ha_objectes_bruts fer
   agafar_objecte;
   mentre hi_ha_bruticia_a_l'objecte fer
      posar_sabó_a_l'objecte;
      passar_el_fregall_per_l'objecte;
   fimentre
   si l'objecte_es_un_plat
      llavors posar_objecte_a_la_pila_de_plats;
      sinó posar_objecte_a_la_pila_genèrica;
   fisi
fimentre
```

Oi que és més llegible i comprensible que la versió no estructurada?

Avantatges de la programació estructurada

Les principals avantatges de la programació estructurada són:

- Els programes són més fàcils d'entendre
- Es redueix la complexitat de les proves a efectuar
- Augmenta la productivitat del programador
- Els programes queden millor documentats internament
- Es redueix els costos de manteniment (modificacions a efectuar)

Inconvenients de la programació estructurada

El principal inconvenient d'aquest tipus de programació és que s'obté un únic bloc de programa, el qual pot esdevenir molt gran, fet que provoca una dificultosa manipulació. Però això es soluciona amb la programació modular.

1.4.2. Programació modular

La programació modular consisteix en dissenyar el programa per parts, anomenades mòduls. Un programa dissenyat de forma modular conté un mòdul principal que coordina les crides a la resta de mòduls. A la seva vegada, cada mòdul té un programa que pot cridar a altres mòduls. Els diferents programes de cada mòdul estan desenvolupats utilitzant la programació estructurada.

1.4.3. Altres tipus de programació

L'objectiu d'aquest crèdit és la programació estructurada i modular, tipus de programació bàsica per a poder abordar qualsevol altre tipus de programació. Enumerem, però, a continuació, altres tipus de programació importants, amb una breu descripció en la que apareixeran termes desconeguts i que no podem abordar en aquest moment donat que estem a les beceroles del coneixement de la programació informàtica, però que és important que comencin a sonar.

1) Programació orientada a objectes

Aquest tipus de programació apareix com l'evolució lògica de la programació estructurada i modular i és una tècnica que augmenta considerablement la velocitat de desenvolupament dels programes gràcies a la reutilització de codi (objectes).

L'element principal en aquesta programació és l'objecte, que podem veure com un compartiment que conté dades i el codi per a gestionar-les. Altres característiques molt potents de la programació orientada a objectes són el polimorfisme i l'herència.

2) Programació concurrent

Aquest tipus de programació s'utilitza quan hem de realitzar vàries accions a la vegada. S'acostuma a utilitzar per a controlar els accessos d'usuaris i de programes a un recurs de forma simultània.

3) Programació funcional

És un tipus de programació caracteritzada per permetre la declaració i crida de funcions dins d'altres funcions.

4) Programació lògica

S'acostuma a utilitzar en intel·ligència artificial i es basa en el càlcul de predicats, teoria matemàtica que permet aconseguir que un ordinador pugui donar solucions intel·ligents en base a fets i regles lògiques.

1.5. Llenguatges de programació.

En general, un llenguatge és un conjunt sistemàtic de regles per a comunicar idees, en base a un conjunt de símbols. A nosaltres ens interessa, ara mateix, els llenguatges de programació.

> Un llenguatge de programació és una notació formal per descriure algorismes que han de ser executats en un ordinador.

Els llenguatges de programació són utilitzat pels programadors per a expressar un procés mitjançant el qual un ordinador resoldrà un problema.

Un llenguatge de programació permet al programador especificar de forma precisa sobre quines dades un ordinador ha d'operar, com han de ser emmagatzemades i transmeses i quines accions ha de prendre sota una variada gama de circumstàncies.

Tota notació formal té dos components: sintaxis i semàntica.

a) La sintaxis és un conjunt de regles formals que especifiquen la composició dels programes en base a un conjunt de lletres, dígits i altres caràcters.

Així, exemples de regles sintàctiques poden ser:

- Cada parèntesi obert ha de portar el corresponent parèntesi tancat en les expressions aritmètiques.
- Tota instrucció ha de finalitzar amb el símbol "punt i coma".

b) La semàntica especifica el significat de qualsevol programa sintàcticament vàlid, escrit en un determinat llenguatge de programació. Les regles semàntiques permeten traduir els programes al codi màquina per a que siguin executats per l'ordinador.

Així, per exemple, les regles semàntiques indicaran que una instrucció sintàcticament correcta com la següent

```
mentre hi_hagi_objectes_bruts fer
   agafar_objecte_brut;
   netejar_objecte_brut;
fimentre
```

consisteix en la repetició ininterrompuda del conjunt format per les dues accions `agafar_objecte_brut` i `netejar_objecte_brut` sempre i quan `hi_hagi_objectes_bruts`.

1.5.1. Classificació dels llenguatges de programació

Al llarg de la història de la programació informàtica han aparegut diversos llenguatges de programació pensats, cadascun d'ells, per a cobrir unes determinades necessitats i per a donar servei a diferents tipus de programació i adequats a les arquitectures d'ordinadors del moment.

Es pot establir diferents classificacions dels llenguatges de programació:

1) Segons el nivell de comunicació programador-màquina

Aquest punt de vista es refereix a la proximitat entre el llenguatge utilitzat pel programador i el llenguatge que entén la màquina. Tenim:

a) Llenguatge màquina
b) Llenguatges assembladors: *Assembler*
c) Llenguatges imperatius d'alt nivell: *Fortran*, *Pascal*, *C*, *C++*, *Ada*,
d) Llenguatges declaratius d'alt nivell: *Prolog*, *Lisp*

Ja som coneixedors de la distinció entre llenguatge màquina, llenguatge assemblador i llenguatge d'alt nivell. Ens ve de nou, però, la distinció, dins els llenguatges d'alt nivell, entre llenguatges imperatius i llenguatges declaratius.

> Els llenguatges imperatius es caracteritzen per indicar com fer les coses mentre que els llenguatges declaratius es caracteritzen per indicar què cal fer.

Els llenguatges declaratius són molt propers a les matemàtiques i a la lògica mentre que els llenguatges imperatius són més propers al raonament humà.

Com a exemple podem pensar en un programa que resolgui el càlcul del número factorial d'un valor enter no negatiu.

Un llenguatge declaratiu podria resoldre el problema dient:

```
factorial(0) = 1
factorial(n) = n*factorial(n-1)
```

És important notar que l'ordre de les instruccions és fonamental, doncs:

- la primera instrucció possibilita que el procés finalitzi
- la segona instrucció possibilita que per a un número superior a 0 (si fos zero ja ho resol la primera instrucció) es torni a repetir el procés.

En canvi, un llenguatge imperatiu resoldria el problema dient com s'ha d'efectuar el càlcul de forma propera al raonament humà:

```
anar multiplicant 1 * 2 * 3 * ... fins arribar a n
```

2) Segons el domini d'aplicació

Sota aquest punt de vista tenim:

a) Llenguatges genèrics (per qualsevol àmbit): *C, C++, Ada, Java*
b) Llenguatges científics: *Fortran, APL, Pascal*
c) Llenguatges comercials: *Cobol, Visual Basic*
d) Llenguatge de bases de dades: *T-SQL, PL/SQL*
e) Llenguatge d'intel·ligència: *Prolog, Lisp*

3) Segons el tipus de programació:

Sota aquest punt de vista tenim:

a) Llenguatges imperatius: *C, Pascal, Fortran*
b) Llenguatges orientats a objectes: *C++, Simula, Smalltalk, Eiffel*
c) Llenguatges funcionals: *Lisp, Scheme, ML, Haskell*
d) Llenguatges de programació lògica: *Prolog*
e) Llenguatges concurrents: *Ada*

Llenguatge ADA

Aquest llenguatge apareix al 1983 i el seu nom és en honor de Lady Augusta Ada Byron (1815-51), filla del poeta anglès Byron, considerada la primera dona programadora d'ordinadors degut al seu treball en la màquina de Babbage.

4) Segons el desenvolupament històric

Sota aquest punt de vista tenim:

a) Llenguatges de 1a. generació: *Fortran* (la versió apareguda al 1957)
b) Llenguatges de 2a. generació: *Algol 60*
c) Llenguatges de 3a. generació: *Pascal, C*
d) Llenguatges posteriors a 3a. generació:
- Llenguatges orientats a objectes: *Java, C++*
- Entorns visuals: *Java, Visual Basic, Visual C, Borland C*

S'acostuma a considerar llenguatges de 3a. generació aquells que donen ple suport a la programació estructurada i modular. La majoria d'ells permeten gestionar sistemes gestors de fitxers per emmagatzemar les dades.

Posterior als llenguatges de 3a. generació van anar apareixent llenguatges especialitzats en entorns visuals i/o en la orientació a objectes.

Cal esmentar també l'existència dels anomenats entorns de 4a. generació que corresponen a sistemes gestors de bases de dades que incorporen diferents llenguatges per a tractaments especialitzats (llenguatge per a tractament de pantalles, llenguatge per a tractament d'informes, llenguatge per a tractament de menús,...)

1.5.2. Entorns integrats de programació

Es parla d'entorn integrat de programació quan el programador treballa amb una eina que li permet editar els programes fonts, invocar els programes traductors, muntadors i depuradors i, fins i tot, executar els programes resultants.

O sigui, un entorn integrat de programació no és altra cosa que l'acumulació de les diferents eines de programació necessàries per a facilitar la feina al programador. Però totes les eines es poden utilitzar directament des del sistema operatiu amb la sintaxi adequada. És a dir,

es pot treballar sense l'eina. Només necessitaríem un editor de textos per a escriure el programa font i podríem aconseguir els programes objecte i executable per mitjà d'ordres directes en l'àmbit del sistema operatiu. Això implica conèixer la sintaxi adequada, i la veritat és que la majoria de programadors treballen sempre en entorns integrats de programació, també anomenats entorns de desenvolupament.

1.6. Introducció al llenguatge C

El llenguatge C és el llenguatge que farem servir en l'ordinador per a l'aprenentatge de la programació estructurada i modular. Introduïm-ne les principals característiques.

1.6.1. Llenguatges C i C++

El llenguatge C va néixer en els laboratoris Bell d'AT&T i ha estat estretament vinculat al sistema operatiu Unix, ja que el desenvolupament d'aquest sistema (per Brian W. Kernighan i Dennis M. Ritchie) i el del llenguatge C (inicialment implantat per Dennis M. Ritchie) varen ser realitzats en paral·lel. El sistema Unix, el compilador de C i la majoria de programes i eines d'Unix varen ser escrits en C sense utilitzar, quasi bé, el llenguatge assemblador.

El llenguatge C

> "C és un llenguatge de programació d'utilització general, caracteritzat per la seva concisió i per tenir un modern flux de control i estructures de dades, així com un ric conjunt d'operadors. No és un llenguatge de "molt alt nivell" ni "gran" i no està especialitzat en cap àrea particular d'aplicació. En canvi, la seva carència de restriccions i la seva generalitat el fan més eficaç i convenient, per a moltes tasques, que altres llenguatges suposadament més potents."
>
> Kernighan, B.W.; Ritchie D.M. (1985). El Lenguaje de programación C. Mèxic: Prentice Hall.

Aquí trobem una primera referència a bibliografia procedent d'Hispanoamèrica. Ja s'ha comentat anteriorment que l'anglès és imprescindible per a tot informàtic, ja que la immensa bibliografia original és escrita en aquest idioma. Hi ha, però, algunes obres traduïdes al castellà i, en temes informàtics, és molt probable que es tracti d'edicions hispanoamericanes. En aquests casos s'ha d'anar amb compte amb alguns termes utilitzats a les traduccions, gens naturals en el nostre entorn.

Al llarg d'aquest crèdit farem referència en més d'una ocasió a aquest tema, indicant el terme utilitzat normalment en les traduccions hispanoamericanes.

Avui en dia està molt de moda el llenguatge C++, resultant de l'evolució del C per tal de permetre la programació orientada a objectes.

C++ és un llenguatge que manté la compatibilitat amb l'estàndard C i que ha adoptat totes les característiques de la programació orientada a objectes. En aquest crèdit programarem amb l'estàndard C, però

l'estudiant podrà fer servir un entorn C o C++ indiferentment, tenint en compte una única consideració: els compiladors són diferents.

El llenguatge C (i també C++) porta el compilador corresponent, que permet generar, a partir del programa font, el programa objecte que és utilitzat pel programa muntador per a aconseguir el programa executable definitiu.

Els programes compiladors de C i C++ són diferents, cosa evident ja que el llenguatge C++ incorpora més conceptes que el llenguatge C. Però, fins i tot, la traducció que efectua el compilador C++ per un programa escrit íntegrament amb C (sense utilitzar conceptes exclusius de C++) és diferent de la traducció efectuada pel compilador C. I aquí heu de posar especial atenció si treballeu en un entorn C++. ‼

Un programa font escrit en llenguatge C ha de tenir un nom i és molt recomanable que la seva extensió sigui `.c`. Un programa font escrit en llenguatge C++ ha de tenir un nom i és molt recomanable que la seva extensió sigui `.cpp`. Hi ha entorns de programació que obliguen la utilització de les extensions .c i .cpp i utilitzen l'extensió per a deduir quin és el compilador a emprar.

En aquest crèdit, tots el codi es desenvolupa amb l'estàndard C. Per tant, tingueu-ho present a l'hora d'anomenar els programes fonts amb la corresponent extensió `.c`.

Ara és el moment d'instal·lar i començar a utilitzar un entorn de programació en C efectuant una lectura detallada de tota la informació que l'entorn aporta. Així, cal tenir clar:

- Com s'escriu un programa font.
- Com es compila un programa font per a obtenir el programa objecte.
- Com es munta un(s) programa(es) objecte per a obtenir el programa executable.
- Com es detecta errors de traducció o muntatge.
- Com s'invoca el programa depurador.

1.6.2. Realització del primer programa

Recordeu el vostre primer programa en pseudocodi? Codifiquem-lo en llenguatge C. El programa era:

```
programa primer_programa és
   escriure ("Hola, món!");
fiprograma
```

Com es codifica l'anterior programa en C? El resultat final seria una cosa semblant a:

```
void main(void)
{
   printf ("Hola, món!");
}
```

‼ Trobareu l'arxiu `u1n1p01.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Tot programa en pseudocodi ha d'anar entre les paraules clau `programa` i `fiprograma`. En llenguatge C, tot programa s'identifica per la paraula clau `main` i el contingut del programa, que constitueix el que anomenarem `bloc`, ha d'anar entre les claus {}. Això només és una primera aproximació de com ha de ser un programa escrit en C.

Observem que la paraula `main` apareix acompanyada de valors per davant i per darrera... De moment us demanem que feu un acte de fe i us ho cregueu. Més endavant ja explicarem el seu significat.

‼ Per a instal·lar diversos entorns de programació en C i poder escriure i executar aquest primer programa, vegeu en la secció "Recursos de continguts" del web d'aquest crèdit "Utilització de l'entorn de programació *Anjuta* per a programar en C" i "Utilització de l'entorn de programació Turbo C++ 1.01"

Veiem, també, que el terme equivalent a `escriure` del pseudocodi és, en C, `printf`. No és l'única manera. Ja en parlarem més endavant.

Ara és el moment ideal per a escriure aquest programa a l'ordinador en l'entorn de programació en C que correspongui per tal d'arribar a assolir el codi executable.

Cal, potser, que ens posem d'acord en la nomenclatura que utilitzarem per a referir-nos a tots els exemples d'aquest llibre. Els programes fonts en llenguatge C els anomenarem: `uXnYpZZ.c`.

Això vol dir:

- Unitat Didàctica X.
- Nucli Activitat Y.
- Programa ZZ (01, 02, 03, ...).

Com ja sabeu, el crèdit està estructurat en unitats didàctiques i aquestes en nuclis d'activitat. Així doncs, el programa anterior l'anomenem `u1n1p01.c`

Sou vosaltres qui escriviu el programa i li doneu nom. En la fase de traducció, l'ordinador genera el programa objecte i l'anomena amb el mateix nom canviant l'extensió (`u1n1p01.obj` en Windows / `u1n1p01.o` en Linux). Per fi, el programa muntador genera el programa executable, seguint la mateixa mecànica de canviar l'extensió (`u1n1p01.exe` en Windows / `u1n1p01` en Linux).
Com observeu, les extensions utilitzades depenen dels sistemes operatius. ‼

És natural anomenar els programes objecte i els programes executables amb el mateix nom (variant-ne l'extensió), ja que així és més fàcil identificar els diferents arxius que tenim en els disquets. Els entorns integrats de programació acostumen a treballar d'aquesta manera, és a dir, el programador dóna nom al programa font, suposem `nom.c`, i l'entorn, quan executa les fases de traducció i muntatge, genera els corresponents compilat i executable mantenint el nom i utilitzant l'extensió que correspongui.

2. Dades i operadors

Ens proposem introduir el concepte dels tipus de dades, bàsic per a la gestió de les dades en els nostres programes, veient-ne diverses classificacions i entrant a fons en el coneixement dels tipus de dades estàtiques simples, en pseudocodi i en el llenguatge C.

2.1. Memòria de l'ordinador

El terme dada indica tota informació que fa servir l'ordinador en les execucions dels programes. (!!)

És a dir, l'execució d'un programa consisteix generalment a gestionar dades que han de residir en alguna part de l'ordinador. Fem ara una simulació del funcionament de l'ordinador quan s'executa un programa. Vosaltres sou l'ordinador i nosaltres som els usuaris. Suposem que teniu un programa que permet preguntar a l'usuari –a través d'un missatge de pantalla– el seu nom i que es queda esperant que l'usuari –mitjançant el teclat– l'introdueixi. Posteriorment l'ordinador ofereix un missatge de benvinguda personalitzant el missatge amb el nom de l'usuari.

Au, som-hi. Nosaltres, que som els usuaris, executem dins vostre (feu d'ordinador) el programa anterior. La seqüència de passos de la situació que es produeix és:

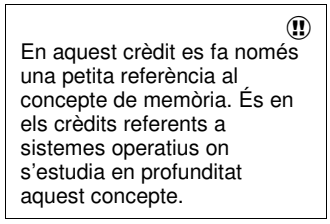
1) Vosaltres pregunteu: "Quin és el teu nom?" i espereu la nostra resposta.
2) Nosaltres responem: "Isidre".
3) Vosaltres dieu: "Benvingut, Isidre".

Hi esteu d'acord? Esperem que sí. Doncs bé, aquest funcionament tan normal en la nostra vida quotidiana és el que hem d'aconseguir en l'ordinador mitjançant el programa informàtic. Hi ha un punt en què cal posar especial atenció: l'ordinador està a l'espera que l'usuari introdueixi el seu nom i, quan succeeix aquest fet, dóna un missatge personalitzat amb el nom que hagi introduït l'usuari.

Quan els éssers humans mantenim un diàleg com l'anterior, la persona que fa d'ordinador i espera la resposta l'emmagatzema en la seva memòria, de manera que, com que disposa d'aquesta informació, la pot utilitzar posteriorment. L'ordinador té un funcionament semblant. Quan rep la resposta de l'usuari, l'ha d'emmagatzemar en algun lloc per a,

posteriorment, donar el missatge personalitzat. Aquest lloc és la memòria.

La memòria de l'ordinador és l'element utilitzat per a emmagatzemar dades.



L'ordinador té, bàsicament, dos tipus de memòries: volàtils i no volàtils.

Una memòria és volàtil si les dades emmagatzemades desapareixen en el moment de desconnectar l'aparell.

Una memòria és no volàtil si les dades enregistrades només desapareixen per una acció específica sol·licitada per l'usuari.

Les dades que tractem en aquest moment són dades que un programa utilitza en execució. L'ordinador les guarda en memòria de tipus volàtil. És a dir, si l'ordinador es desconnecta mentre s'està executant, aquestes dades es perdran. També es perden en finalitzar l'execució del programa si no és que el programa es preocupa d'enregistrar-les en memòria de tipus no volàtil.

Enregistrament de dades

Per a enregistrar les dades que gestiona un programa abans del seu acabament s'utilitzen els arxius en disc.

La memòria d'un ordinador també admet una altra classificació: interna i externa.

La memòria interna (també anomenada central) és memòria que hi ha a la pròpia placa base de l'ordinador. La memòria externa (també anomenada en massa) és la memòria emmagatzemada en altres tipus de suport, bàsicament magnètics i òptics.

Per a entendre'ns, si extrapolem aquests dos tipus de memòria a l'ésser humà, la memòria interna es correspondria amb el cervell, mentre que la memòria externa es correspondria amb els suports externs de què l'ésser humà disposa per a guardar aquells coneixements que no vol (o no pot) mantenir en el seu cervell i per als quals utilitza paper, cintes d'àudio, cintes d'imatge...

De la mateixa manera que la capacitat de la memòria humana és limitada, la memòria interna de l'ordinador també ho és. En canvi, els suports externs de què disposa l'ésser humà per a emmagatzemar coneixements

són pràcticament il·limitats, cosa que també passa amb la memòria externa de l'ordinador. (!!)

La major part de la memòria interna de l'ordinador és volàtil i es fa servir per a emmagatzemar les dades que gestionen els programes. Quan acaba la seva execució, les dades gestionades deixen lliure la memòria per a noves dades que hagin de gestionar altres programes. Només una petita part de la memòria interna és no volàtil i no és modificable pels programes en execució. Allí es guarden les dades necessàries perquè l'ordinador sàpiga fer alguna cosa quan es connecta.

Memòria interna volàtil i no volàtil: RAM i ROM

La part volàtil de la memòria interna s'anomena RAM (Random Access Memory) i és memòria de lectura i escriptura, ja que els programes la poden llegir i hi poden escriure. És com si escrivíssim en un full de paper amb un llapis. La part escrita es pot esborrar i tornar-hi a escriure. En la memòria RAM, no cal esborrar per a tornar a escriure; simplement s'escriu al damunt, igual com succeeix quan enregistrem una cançó en una cinta d'àudio que ja havia enregistrat altres cançons.

La part no volàtil de la memòria interna s'anomena ROM (Read Only Memory) i és només de lectura. Els programes només la poden llegir. No hi poden escriure. Llavors, qui l'ha escrit? És el fabricant de la placa base de l'ordinador qui l'ha "gravat" amb uns mecanismes especials.

La memòria interna és d'accés molt ràpid perquè es troba a la placa base de l'ordinador, de manera semblant a la memòria del nostre cervell, a la qual accedim instantàniament.

En canvi, la memòria externa, constituïda per suports magnètics (discos durs, discos flexibles, cintes magnètiques) i òptics (els actuals discos compactes) és d'accés més lent, de manera semblant als suports externs utilitzats tradicionalment per l'ésser humà (per exemple, el paper).

Les memòries externes són bàsicament de lectura i escriptura. Això no vol dir que no us en pugueu trobar de només lectura, de la mateixa manera que una cinta d'àudio o de vídeo pot estar protegida i no permetre l'enregistrament.

2.2. Variables i constants

Ara que ja sabem on guarda l'ordinador les dades que el programa gestiona, introduirem una classificació d'aquests tipus de dades.

Una dada gestionada en un programa pot ser, segons la durada de la seva existència, de dos tipus: constant o variable.

> Una dada constant és una dada el valor de la qual existeix des de l'inici de l'execució del programa fins a la fi i es manté inalterable.

Continuant amb la comparació entre el funcionament d'un ordinador i de l'ésser humà, exemples de dades constants en les persones serien, sense voler entrar en cap tipus de discussió sobre si el valor d'alguna

podria variar segons la persona, el nom, la data i el lloc de naixement, el sexe i el color d'ulls. Aquestes dades podríem dir que existeixen des del naixement de l'ésser humà i es mantenen inalterables fins a la seva mort.

En altres entorns també tenim exemples de constants:

- els nombres π i e en matemàtiques,
- el nom fiscal i el codi d'identificació fiscal d'una empresa (podríem discutir-ho).

Una dada variable és una dada que ocupa una zona de memòria i el valor que representa pot anar canviant durant l'execució del programa.

Exemples de dades variables, en l'ésser humà, són el nom de la parella; les dates de casament, separació, divorci, etc.; el pes; l'alçada; el nivell d'estudis, etc.

Dades variables i constants en les cultures

Podem exceptuar algunes tribus indígenes en les quals la persona ja té la parella assignada en el moment del seu naixement. Seria una constant si no pogués tenir una altra parella a la seva vida, però seria variable si li fossin permeses altres parelles.

Per a algunes persones, alguna d'aquestes dades es pot mantenir inalterable des del moment de la seva aparició (una persona pot tenir una parella estable tota la seva vida), però és una variable i no una constant, ja que no ha existit durant tota la vida de la persona (la parella apareix en un determinat moment). En canvi, altres dades com el pes o l'alçada segur que no donen lloc a discussió: tothom les considera variables.

I per què fem servir constants? Evidentment podríem passar sense constants i treballar només amb variables mentre tinguéssim la precaució de no modificar el seu valor al llarg de l'execució. La resposta a la pregunta està en el fet que treballant amb constants podem facilitar la modificació de programes en determinades ocasions.

Utilitat de les constants

Suposem que un programa gestiona classes d'una escola amb una capacitat màxima de vint-i-cinc alumnes per aula. Segur que dins del programa el valor vint-i-cinc alumnes apareix en un munt de llocs. Si alguna vegada s'ha de refer el programa per causa d'una modificació de la capacitat màxima, caldrà cercar en el programa tots els llocs on apareix el valor 25, comprovar que correspon a la capacitat màxima (podríem tenir un valor 25 que correspongués a un altre significat) i modificar-lo adequadament. En canvi, si aquest valor el tenim declarat com una constant CAPACITAT=25 i la utilitzem al llarg del programa, només haurem de canviar la línia de la seva declaració.

Ja hem dit que una constant existeix durant tota l'execució del programa i és inalterable. En canvi, una variable pot existir durant tota l'execució del programa (variables estàtiques) o durant una part de l'execució i, fins i tot,

tenir més o menys capacitat segons la necessitat de valors que s'hagin d'emmagatzemar (variables dinàmiques). Aquesta situació ens permet diferenciar els tipus de variables, segons el moment temporal en què existeixen.

Una variable estàtica existeix durant tota l'execució del programa. Té una capacitat fixa i els valors que emmagatzema poden variar.

Una variable dinàmica es crea i es destrueix quan cal durant l'execució del programa. Té capacitat decidida en temps d'execució i els valors que emmagatzema poden variar.

Per a un bon tractament de variables dinàmiques, cal conèixer a fons el tractament de variables estàtiques i, per tant, aquest és el nostre primer objectiu.

Podem encara classificar les variables segons la capacitat d'emmagatzematge.

Una variable simple és una variable que té capacitat per a emmagatzemar un únic valor.

Una variable composta és una variable que té capacitat per a emmagatzemar diversos valors.

Es produeixen exemples de variables simples quan el programa ha de gestionar dades com les següents:

- edat de la persona,
- nom de la persona (discutible, ja que el nom és un conjunt de caràcters),
- nombre d'alumnes d'una classe,
- capacitat en litres d'un dipòsit.

Es produeixen exemples de variables compostes quan un programa ha de gestionar dades com les següents:

- notes dels alumnes matriculats en un crèdit,
- dades d'una persona de manera global: el nom, el sexe, la data de naixement, el pes, l'alçada, el DNI, la professió, l'adreça, el codi postal, la població, etc.

Ara que ja tenim les dades classificades, les farem servir en els nostres programes.

> Tota dada que fem servir en un programa ha d'haver estat declarada pel programador.

I com es fa, això?

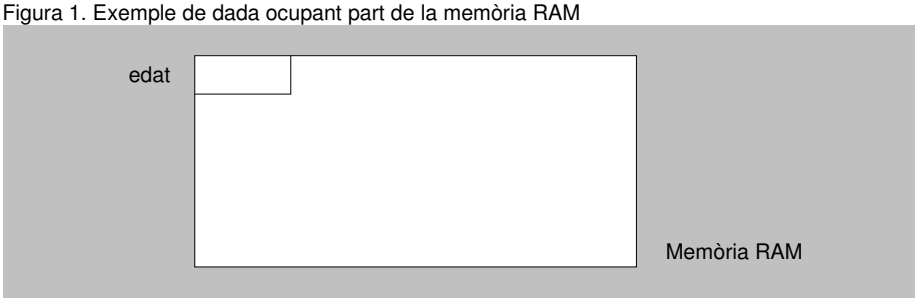
En pseudocodi les dades que s'han de gestionar les declarem en una zona específica per a tal finalitat, situada com es veu a continuació:

```
programa   <nom programa> és
const <nom_constant_1> = <valor_1>;
      <nom_constant_2> = <valor_2>;
      ...
ficonst
var   <nom_variable_1> : <tipus_variable_1>;
     <nom_variable_2> : <tipus_variable_2>;
     ...
fivar
<instruccions del programa>
fiprograma
```

Tal com heu intuït, les constants es declaren en una zona delimitada per les paraules reservades `const` i `ficonst` i les variables, de manera similar, en una zona delimitada per `var` i `fivar`.

També veieu que les constants i les variables tenen un `nom`, el qual farem servir dins el programa per a referenciar la zona de memòria que conté el valor emmagatzemat. És a dir, si tenim declarada una dada (constant o variable) de nom `edat`, quan l'ordinador comenci l'execució del programa destinarà una zona de la memòria a aquesta dada, com s'exemplifica a la figura 1.

Figura 1. Exemple de dada ocupant part de la memòria RAM



El terme edat és el nom de la dada, que permet al programador fer referència, en el programa, a la zona de memòria assignada. Aquesta zona no és la mateixa a cada execució del programa; la posició que ocupa la decideix el sistema operatiu quan comença cada execució. El que no

variarà al llarg de les diferents execucions és la capacitat i, per tant, l'espai que ocupa.

El nom d'una dada és la manera que té el programador per referenciar la zona de memòria que el sistema operatiu assignarà a cada execució del programa.

Cada llenguatge informàtic té unes determinades normes que fan referència a com pot ser aquest nom que decideix el programador. No difereixen gaire. En pseudocodi considerarem que: (!!)

Particularitats del pseudocodi

En pseudocodi, el llenguatge informàtic no pot contenir comes, punts, punts i comes, dos punts, etc.

- pot ser una seqüència de lletres i dígits de qualsevol longitud;
- no pot contenir signes de puntuació;
- no pot tenir espais en blanc;
- no pot ser cap de les paraules reservades del pseudocodi;
- no hi pot haver, en principi, dades amb el mateix nom;
- no hi ha distinció entre les majúscules i les minúscules.

Altres opcions

En alguns llenguatges, com el C, hi ha distinció entre majúscules i minúscules, de manera que dades amb noms EDAT, Edat, edat, eDat, etc. són considerades diferents. En tal cas es diu que el llenguatge és case sensitive.

El sentit comú indica que el `nom` d'una dada hauria d'estar pensat de manera que tingui a veure amb el valor que hi volem emmagatzemar. Així, a l'exemple `edat`, cal suposar que es tracta d'una dada per a contenir l'edat d'alguna cosa. No seria gens lògic utilitzar aquest terme per a emmagatzemar el sou d'una persona.

Què cal fer si necessitem tenir dues dades de tipus edat, una per a un home i una altra per a una dona? En tal cas caldrà declarar dues dades i podríem pensar en diferents possibilitats:

- `edat1` i `edat2`: caldrà tenir molt clar quina es refereix a l'home i quina a la dona.
- `edath` i `edatd`: ja és més entenedor, però encara pot portar a confusió.
- `edat home` i `edat dona`: és el més clar; no hi ha dubtes.

Subratllat

L'argot informàtic acostuma a fer servir el terme anglès underline ('subratllat')

La darrera possibilitat és errònia perquè conté espais en blanc. Per solucionar això, s'acostuma a utilitzar el símbol '_' (subratllat) per a simular els espais en blanc. Així doncs, la darrera possibilitat seria `edat_home` i `edat_dona`.

Hi ha algun conveni específic referent als noms de les dades? En principi el programador té total llibertat per a decidir-ne el nom. Hi ha, però, certs convenis notacionals, els quals se solen fer servir en alguns llenguatges concrets, però que es poden generalitzar en altres entorns. Així, en pseudocodi escriurem les constants amb lletra majúscula i les variables amb lletra minúscula.

2.3. Tipus de dades estàtiques simples.

Suposem que volem que `edat` sigui una variable que ha de contenir un valor que variarà al llarg del programa. Hauríem de declarar-la a la corresponent zona de variables:

```
var ...
    edat : <tipus de variable>;
    ...
fivar
```

I què vol dir <tipus de variable>?

> El tipus d'una dada permet definir el conjunt de valors que pot prendre la dada.

És clar que la nostra variable `edat` ha de contenir valors numèrics de tipus enter. No hem de permetre que `edat` contingui per valors una paraula, una data, etc. És discutible, però, si hem de permetre o no valors decimals per poder representar situacions semblants a "té dotze anys i mig".

Tots els programes gestionen informació que es representa en forma de dades, però no totes les dades tenen la mateixa naturalesa. Aquí apareix el concepte de tipus.

Tipus de dades

Per exemple, "27-1-1958" és una dada i "Teresa" és una altra dada, però evidentment no són del mateix tipus. Això es pot comprovar si intentem fer operacions entre les dues. Oi que no té cap sentit intentar sumar-les?

Per tot això apareix la necessitat de classificar les dades en uns certs tipus.

El tipus de dada és un conjunt de valors que porta associades unes operacions que es poden realitzar sobre aquestes dades i que tenen una representació interna determinada per a l'ordinador, la qual cosa depèn del sistema operatiu utilitzat.

Els tipus de dades que es pot utilitzar en un programa i la informació que podem representar amb aquelles depèn del sistema operatiu i del llenguatge de programació emprat. De tota manera hi ha un gran acord en els tipus de dades necessaris i la majoria dels llenguatges els incorporen.

El concepte de tipus de dada és bàsic en programació, ja que l'ordinador assigna a cada dada declarada en el programa una determinada zona de memòria on emmagatzema el valor amb la representació interna que correspongui. (!!)

Comencem el coneixement dels tipus de dades amb les possibilitats existents en l'àmbit dels tipus de dades estàtiques simples. A continuació en fem la classificació que cal tenir en compte a l'hora de dissenyar els algorismes en pseudocodi. Quan codifiquem el programa pseudocodi en un llenguatge informàtic, caldrà conèixer les possibilitats de tipus de dades que aquest llenguatge informàtic aporta.

Passem, doncs, a la classificació:

1) Definides: les porten incorporades els llenguatges de programació; segons sigui la naturalesa del valor que representen es poden classificar en:

a) Numèriques: per a representar valors numèrics que es puguin utilitzar en operacions aritmètiques. Aquí trobem els tipus següents:

- Enter: valors positius i negatius sense decimals (el zero també).
- Natural: valors positius sense decimals (el zero també).
- Real: qualsevol valor numèric.

b) Caràcter: per a representar els caràcters, entenent per caràcter qualsevol lletra, xifra, signe de puntuació, símbol matemàtic o de qualsevol altre tipus que es pugui utilitzar.

c) Lògiques: per a representar els dos valors lògics cert i fals.

2) Definibles: aquelles en les quals el conjunt de valors possibles el pot definir el programador. Hi trobem dades dels tipus següents:

a) Enumerades: definides pel conjunt de valors que utilitza (enumeració de tots els valors possibles) i les operacions que s'hi poden efectuar.

b) Subrangs: definides com un subconjunt de valors dels enters, naturals, caràcters o d'un tipus enumerat.

En els apartats següents abordarem amb deteniment cadascun d'aquests tipus i les operacions que es poden desenvolupar amb les dades que emmagatzemen.

La notació hongaresa

En el món de la programació hi ha alguns convenis notacionals per al nom de les dades. Ara que hem introduït el concepte de tipus, podem parlar d'una notació utilitzada en certs entorns.

La notació hongaresa consisteix a iniciar el nom de totes les variables amb una lletra que indica el seu tipus (n per al tipus natural, e per al tipus enter, r per al tipus real, c per al tipus caràcter, l per al tipus lògic, etc.) seguit del nom de la variable en minúscules excepte la primera lletra, que va en majúscula. Exemples: `nEdat`, `lEsCasat`, `cSexe`, `rSou`, etc.

2.4. Dades de tipus numèric

Són les dades que permeten emmagatzemar valors numèrics. N'hi ha de diferents tipus a causa dels diferents conjunts de nombres existents. En pseudocodi considerarem els tres tipus bàsics següents: enter, natural i real.

Conjunts de nombres

En matemàtiques, els conjunts de nombres són N, `Z`, Q, `R`, etc.

Abans, però, cal tenir clar que cada combinació de sistema operatiu i llenguatge de programació utilitza una determinada quantitat de memòria per a destinar-la a una dada d'un determinat tipus en temps d'execució. Expliquem-nos.

Suposem un sistema operatiu S, on s'executarà un programa P codificat en un llenguatge de programació L. Suposem que el programa gestiona una dada D de tipus T. Doncs bé, en executar-se el programa P, el sistema operatiu S destina una zona de memòria (referenciada pel nom de D) amb la capacitat fixada segons el tipus T pel llenguatge L en el sistema S. Suposem que aquesta zona ocupa N_1 octets. Doncs bé, el mateix programa executable aconseguit per una combinació diferent de llenguatge i sistema, tot i utilitzar un tipus de dada equivalent pot tenir per a la dada D una zona de memòria assignada de N_2 octets diferent de N_1.

(!!) En aquests moments ja heu d'estar al corrent de la representació interna de les dades, tema que es tracta en els crèdits referents a sistemes operatius.

Aquest fet és molt important, ja que l'ordinador, amb un espai finit, només pot representar una quantitat finita de dades. Penseu en la típica situació que us trobeu quan empleneu un determinat formulari en què hi ha caselles per a situar les vostres dades personals. Oi que en aquestes

caselles no disposeu d'espai il·limitat? Doncs bé, els espais assignats per les dades en memòria tenen també unes capacitats fixades i, per tant, els valors que s'hi han d'emmagatzemar són finits. Això és un fet que el programador ha de tenir molt clar i, per a fer-ho més problemàtic, depèn del llenguatge i del sistema operatiu utilitzats. També cal dir que hi ha una gran normalització en aquest aspecte i que no ens trobarem amb gaires diferències, però sempre que es codifiqui un programa en un determinat llenguatge, el programador ha de conèixer els tipus de dades que el llenguatge permet i les implicacions de cadascun d'aquests.

Tot llenguatge de programació té un manual de referència, on hi ha d'haver l'explicació detallada del llenguatge en qüestió. En versions antigues acostuma a ser imprès en paper, però les versions actuals el solen incorporar en el sistema d'ajuda de l'entorn de programació. En tal cas parlem de "manual en línia" (en argot informàtic anglès, on-line).

En pseudocodi fugirem d'aquestes consideracions i suposarem que una dada d'un determinat tipus permet emmagatzemar qualsevol valor d'aquell tipus. En canvi, quan apliquem tot això al llenguatge C veureu que caldrà tenir en compte totes les limitacions.

2.4.1. Dades de tipus enter

Les dades tipus enter permeten emmagatzemar un subconjunt finit (per la capacitat fixa de memòria assignada) dels nombres enters (conjunt matemàtic $\mathbb{Z}$).

És a dir, permeten guardar valors com 0, 1, 2, 3, ..., –1, –2, –3, ... La capacitat finita provoca que hi hagi un límit per la banda positiva i un límit per la banda negativa.

2.4.2. Dades de tipus natural

Permeten emmagatzemar un subconjunt finit (per la capacitat fixa de memòria assignada) dels nombres naturals (conjunt matemàtic $\mathbb{N}$).

És a dir, permeten guardar valors com 0, 1, 2, 3, ... La capacitat finita provoca que hi hagi un límit.

2.4.3. Dades de tipus real

Permeten emmagatzemar un subconjunt finit (per la capacitat fixa de memòria assignada) dels nombres reals (conjunt matemàtic $\mathbb{R}$).

És a dir, permeten guardar valors com 0, 1, –1, 3.5, 4.333, –7.9, ... La capacitat finita provoca que hi hagi un límit per les bandes positiva i negativa i, a més, que els valors reals que tenen un conjunt infinit (o molt gran) de nombres decimals no poden tenir la seva representació exacta. Per això, quan es parla de representació de nombres reals caldrà especificar la precisió amb què es treballa. En pseudocodi no ens ha de preocupar; sí, en canvi, en la codificació en un llenguatge en concret.

2.4.4. Operadors sobre el tipus numèric

Ja tenim les dades numèriques tipificades, però poca cosa podríem fer sense poder-hi operar. Necessitem, doncs, uns operadors numèrics. Aquest concepte també depèn de cada llenguatge de programació. En pseudocodi considerarem els presentats a la taula 1.

Possibilitats dels operadors numèrics

Estem parlant d'operadors que ens permeten efectuar operacions internes, és a dir, operem amb dades numèriques per a obtenir una altra dada numèrica. Veurem, més endavant, la possibilitat d'operar amb dades d'un tipus per a obtenir dades d'un altre tipus. En aquest cas, estarem en operadors externs.

Taula 1. Operadors numèrics en pseudocodi

Operador +	Efectua la suma de valors numèrics.
Operador –	Efectua la diferència de valors numèrics.
Operador *	Efectua el producte de valors numèrics.
Operador /	Efectua la divisió real de valors numèrics (donant resultat amb decimals).
Operador div	Calcula el quocient enter de la divisió entera de valors numèrics enters (dóna el quocient sense "baixar" decimals).
Operador mod	Calcula el residu de la divisió entera de valors numèrics enters (residu calculat sense haver "baixat" decimals).

Cal destacar la importància del tipus del resultat en funció dels tipus dels operands. En això, hem d'assumir la regla següent: quan operem amb dos valors numèrics de diferent tipus, el resultat serà del tipus més global. És a dir:

- Si en sumem un de natural i un de real, el resultat serà real.
- Si en multipliquem un de natural i un d'enter, el resultat serà enter.
- I si en dividim (amb div) un de natural i un de real? Error, ja que la divisió entera exigeix que els operands siguin enters (els naturals són un subconjunt dels enters i no causen l'error; aquest és produït per l'operand real).

2.4.5. Declaració de variables i constants de tipus numèric

Una vegada tenim tots els ingredients necessaris, podem passar a declarar variables i constants en els nostres programes, com es veu a les següents línies de pseudocodi.

```
const   N = 25; /* constant assumida com a natural i amb valor 25*/
        R = 2.5; /* constant assumida com a real i amb valor 2.5*/
        Z = -5; /* constant assumida com a enter i amb valor -5*/
ficonst

var     edat : natural; /* variable declarada com a natural*/
        saldo_ptes : enter;   /* variable declarada com a enter*/
        saldo_euro : real; /* variable declarada com a real*/
fivar
```

Comentem ara les línies de pseudocodi anteriors. Veureu com en traurem força suc.

En primer lloc, observeu que les dades constants no les declarem de cap tipus en concret. Ja sabem que les dades constants conserven el seu valor inalterable durant tota l'execució del programa. Per tant, cal donar valor en el moment en què es declaren –tal i com es veu a les anteriors línies de pseudocodi–. El programa traductor, avaluant la naturalesa del valor escrit pel programador, assigna el tipus que pertoqui.

Terminologia

En terminologia informàtica, el punt decimal sempre és un punt, mai una coma. Una altra cosa és parlar de com apareixen en pantalla o en paper els valors numèrics reals (això es parametritza en funció de les preferències de l'usuari).

El valor 25 l'assumeix com a natural o com a enter? No us preocupeu. Penseu que sigui quina sigui la decisió el valor serà inalterable i per tant no ens ha de preocupar.

Si volem exigir al programa traductor que una constant amb valor 5 sigui assumida com a real, escriurem 5. (dígit 5 seguit del punt decimal).

I les variables? Què observeu de diferent respecte de les constants? La resposta ens porta a una altra pregunta: quin valor guarden les variables quan es comença l'execució del programa?

Les variables es declaren d'un tipus i no podem fer cap suposició respecte del valor inicial que emmagatzemen. Hem de considerar el seu valor inicial com a "valor escombraria" o "valor porqueria", conseqüència de la interpretació que faci l'ordinador dels valors (zeros i uns) que hi ha en els bits que ocupen.

Hi ha llenguatges que asseguren els valors inicials que prenen les variables. En tal cas, el normal és que les variables numèriques quedin inicialitzades amb el valor zero. Però això no és sempre així.

Per tant, és responsabilitat del programador inicialitzar totes les variables que fa servir en un programa.

Aquesta inicialització no cal fer-la a les primeres instruccions del programa. Es pot fer quan la variable es necessiti per primera vegada.

Segur que ja penseu en la pregunta: i com s'inicialitzen, llavors, les variables? La resposta la trobarem quan coneguem la instrucció d'assignació.

Hi ha llenguatges en què les variables no es declaren i, per tant, no tenen cap tipus assignat. En tal situació una variable s'utilitza quan es necessita i el tipus que pren depèn del valor que emmagatzema. És més, en aquests llenguatges s'acostuma a permetre que la variable pugui canviar de tipus durant l'execució del programa.

A l'apartat "Instrucció d'assignació" podreu veure com es pot assignar valor a una variable en qualsevol punt d'un programa.

Aquest funcionament trenca les bases de la programació. En aquest llibre les considerem un atemptat contra la integritat de la programació. Sabem que hi ha programadors que consideren que aquesta manera de funcionar és millor perquè s'estalvien haver d'escriure tota la part corresponent a la declaració de les variables.

Quins raonaments es pot aportar per justificar la declaració de les variables?

- Obliga que el programador pensi, abans de començar a programar, en totes les variables que necessitarà en el programa (obliga a analitzar i dissenyar millor l'algorisme).
- Davant qualsevol modificació que calgui efectuar en el programa, hi ha una zona on podem trobar la declaració de totes les variables amb, fins i tot, comentaris explicatius relatius al significat i a la utilització dels valors que emmagatzemen.

Per acabar aquest apartat, comentarem la possibilitat d'incorporar en els programes font, comentaris que ajudin a comprendre'ls; en el cas que ens ocupa, comentaris per a entendre la utilització de les constants i variables.

La immensa majoria de llenguatges ofereixen la possibilitat d'incloure comentaris en el programa font, els quals són ignorats pel programa traductor. En pseudocodi també farem servir aquesta possibilitat. Considerarem que qualsevol text escrit entre els símbols /* i */ és un comentari, tal com podem observar en els exemples anteriors.

2.5. Dades de tipus caràcter

Aquest tipus de dada el trobareu en tots els llenguatges de programació. Podríem dir que és el tipus de dada més "internacional".

Caràcter

Pot ser una lletra, una xifra, un símbol matemàtic, un signe de puntuació, etc.

Entenem per caràcter qualsevol lletra, xifra, signe de puntuació, símbol matemàtic o de qualsevol altre tipus que es pugui utilitzar en les representacions de les dades.

En pseudocodi, els valors de tipus caràcter els notarem entre cometes simples (‘), cosa que permet diferenciar-los dels noms de les possibles variables.

Tots els llenguatges fan servir aquest conveni o l’alternatiu, que consisteix a utilitzar cometes dobles (“). Fins i tot alguns llenguatges permeten els dos tipus de conveni. En pseudocodi farem servir cometes simples. Les cometes dobles s’utilitzen per a un altre tipus de dada (cadena) que encara no coneixem.

Atenció! No s’ha de confondre mai un caràcter dígit (per exemple ‘5’) amb un número d’una xifra (número 5 en l’exemple que ens ocupa). En el primer cas no és res més que un símbol. En el segon cas és un valor numèric que es pot utilitzar en càlculs matemàtics. ‼

2.5.1. Operadors sobre el tipus caràcter

En el tipus caràcter no considerarem, per ara, l’existència de cap operador intern, és a dir, els caràcters no es poden operar per a obtenir un caràcter com a resultat.

2.5.2. Declaració de variables i constants de tipus caràcter

De manera semblant a les dades numèriques, declarem constants i variables de tipus caràcter.

```
const   lletra = 'A';
        /*constant assumida com a caràcter i amb valor 'A'*/
ficonst

var  vocal : caràcter; /*variable declarada com a caràcter*/
fivar
```

Pel que fa a la inicialització de les variables de tipus caràcter, hi ha els mateixos problemes que en les numèriques. Això és responsabilitat del programador. En alguns llenguatges s’assegura la inicialització de les variables caràcter amb un espai en blanc (el qual també és un caràcter).

2.5.3. Representació de caràcters

Quants caràcters podem representar? Com molt bé sabeu, l'ordinador sempre té activa una taula de caràcters formada per 256 símbols. Per tant, els caràcters representables són aquest conjunt, els quals segueixen un determinat ordre i estan numerats començant per 0 i fins a 255.

El codi ASCII

A l'hora de decidir quins són els 255 caràcters representables i quin ordre segueixen, s'utilitzen diferents codis establerts, entre els quals destaquen el codi ASCII i el codi EBCDIC.

El codi ASCII és el més estès, sobretot per la gran proliferació d'ordinadors personals (anomenats, en anglès, Personal Computer –PC–). El codi ASCII inicial representava 128 caràcters en lloc dels 256. Quan es van necessitar caràcters especials per als diferents idiomes, es va ampliar amb 128 caràcters més, i es va constituir el codi ASCII estès. Com que hi ha diferents extensions, cal especificar sempre quina és la taula ASCII carregada a l'ordinador (conjunt de caràcters representables).

2.6. Dades de tipus lògic

Dada lògica

En certs entorns de programació, les dades lògiques s'anomenen dades booleanes.

Una dada lògica només pot prendre dos valors: cert o fals.

No tots els llenguatges incorporen aquest tipus de dada. Alguns fan servir altres tipus de dades per a aconseguir el servei que faciliten les dades lògiques.

Lògica

La lògica és la ciència del raonament, de les lleis del pensament, i és imprescindible per a treballar en programació informàtica.

Quina utilitat tenen? En principi la de representar un dels dos valors, cert o fals, però aquest raonament és incorrecte, ja que això també podríem fer-ho amb una variable de tipus caràcter, enregistrant els valors 'C' o 'F'. La gràcia d'aquest tipus de dada és emmagatzemar els valors utilitzats en lògica, els quals són únicament dos, ni més ni menys, i per als quals hi ha unes operacions ben definides.

2.6.1. Operadors sobre el tipus lògic

Operació binària i unària

Una operació binària és aquella que actua sobre dos operands com, per exemple, la suma o la resta. La majoria d'operacions són binàries. En canvi, una operació unària actua sobre un únic operand com, per exemple, l'arrel quadrada, la potència, etc.

Els operadors necessaris en aquest tipus de dada són els que permeten executar les operacions de la lògica: la conjunció (**i**), la disjunció (**o**) i la negació (**no**).

La conjunció és l'operació lògica binària que pren com a resultat el valor cert quan els dos operands tenen valor cert i fals en qualsevol altre cas.

La disjunció (també anomenada disjunció no exclusiva) és l'operació lògica binària que pren com a resultat el valor cert quan algun dels

dos operands (o ambdós) té el valor cert i fals en qualsevol altre cas.

La disjunció exclusiva és l'operació lògica binària que pren com a resultat el valor cert quan només un dels dos operands té el valor cert i fals en qualsevol altre cas.

La negació és l'operació lògica unària que canvia el valor lògic de l'operand.

Els llenguatges fan servir diferents símbols per a representar els operadors corresponents. La taula 2 ens ho mostra.

Taula 2. Símbols per a representar els operadors lògics

Operació	Lògica	Pseudocodi	Llenguatges
Conjunció	$\wedge$	i	and
Disjunció	$\vee$	o	or
Disjunció	XOR	xor	xor
Negació	$\neg$	no	not

Per a entendre millor el comportament dels operadors lògics ens poden ajudar les taules de veritat.

Les taules de veritat són un procés que ens permeten esbrinar, amb claredat i fiabilitat, tots els possibles resultats d'una operació lògica a partir dels valors dels operands.

Taules de veritat

En general, si tenim n dades lògiques, la representació en taules de veritat haurà de considerar "2 elevat a n" situacions.

Això és factible perquè no hi ha un gran nombre de valors inicials possibles. Vegem-ho.

Suposem que tenim dues variables o constants lògiques, A i B, amb les quals volem operar fent servir les operacions binàries conjunció, disjunció i disjunció exclusiva. A i B només poden tenir dos valors; per tant, la combinació de les dues dades només pot considerar quatre possibles situacions, és a dir, un nombre petit de situacions fàcilment representables, com queda palès a la taula 3.

Per al cas de la negació també podem utilitzar una taula de veritat (taula 4), però en aquest cas, s'aplica a una única dada lògica, ja que la negació és una operació unària.

Taula 3. Taula de veritat per a les operacions lògiques conjunció, disjunció i disjunció exclusiva

A	B	A i B	A o B	A xor B
Cert	Cert	Cert	Cert	Fals
Cert	Fals	Fals	Cert	Cert
Fals	Cert	Fals	Cert	Cert
Fals	Fals	Fals	Fals	Fals

Taula 4. Taula de veritat per a l'operació lògica negació

A	no A
Cert	Fals
Fals	Cert

2.6.2. Declaració de variables i constants de tipus lògic

De manera semblant a la les dades numèriques i caràcter, declarem constants i variables de tipus lògic.

```
const   V = cert;  /*constant assumida com a lògica i amb valor cert*/
ficonst

var   és_vocal : lògic;  /*variable declarada com a lògica*/
fivar
```

Pel que fa a la inicialització de les variables de tipus lògic, hi ha el mateix problema que en la resta de tipus: és responsabilitat del programador. En alguns llenguatges s'assegura la inicialització de les variables lògiques amb el valor fals.

2.6.3. Operadors relacionals

Les operacions conegudes sobre dades numèriques i caràcters són operacions internes, és a dir, operen amb dades d'un tipus per obtenir un resultat del mateix tipus. Introduïm ara els operadors relacionals (operadors de comparació) que operen amb dades d'un mateix tipus (numèric o caràcter) i donen com a resultat una dada de tipus lògic.

> Els operadors de comparació són, com el seu nom indica, operadors que permeten comparar dades.

A tall d'exemple, són operadors que permeten preguntes com les següents:

- L'Oriol té més anys que en Guifré?
- La Maria Teresa és més alta que la Loli?
- Són iguals els dos cotxes?

Totes aquestes qüestions tenen un denominador comú: el resultat és de tipus lògic, ja que només pot prendre dos valors: cert o fals.

Com a representació dels operadors de comparació s'utilitzen els símbols matemàtics als quals tots estem acostumats i que recollim a la taula 5.

Taula 5. Operadors relacionals

Operador	Matemàtica	Pseudocodi	Llenguatges
… menor que …	<	<	<
… menor o igual que …	≤	≤ o <=	<=
… major que …	>	>	>
… major o igual que …	≥	≥ o >=	>=
… igual que …	=	= o ==	= o ==
… diferent que …	≠	≠ o <> o !=	<> o !=

Mentre que en els diferents llenguatges i pseudocodis hi ha unanimitat respecte dels quatre primers operadors, no passa el mateix amb els dos darrers.

La majoria de pseudocodis que trobareu en llibres de programació fan servir:

- Un signe '=' per a l'operador "… igual que …".
- El signe '≠' o la combinació '<>' per a l'operador "… diferent que …".

En canvi, en el material que teniu a les vostres mans, farem servir:

- Dos signes '=' per a l'operador "… igual que …", o sigui, '=='.
- La combinació '!=' per a l'operador "… diferent que …".

El motiu d'aquesta decisió és facilitar-vos l'aprenentatge de la programació en llenguatge C, el qual utilitza la nomenclatura exposada.

Particularitats del llenguatge C

El llenguatge C també incorpora el símbol '=', però té una finalitat diferent a la de l'operador de comparació "… i igual que ...". El llenguatge C no acostuma a queixar-se quan se li introdueix aquest símbol, per la qual cosa molts programadors l'utilitzen equivocadament com a operador de comparació i, en executar el programa, aquest no obté el resultat esperat. Això provoca la desesperació del programador, que està convençut que ha dissenyat i codificat de manera correcta l'algorisme.

Tots teniu clar com l'ordinador compara valors numèrics, ja que coneixeu quina és la relació d'ordre definida en els conjunts numèrics. Ara bé, com actuarà l'ordinador en la comparació de dos caràcters?

Més amunt hem comentat que el conjunt de caràcters consta de 256 elements que estan numerats des de 0 fins a 255. Doncs bé, quan l'ordinador es troba amb dues dades, A i B, de tipus caràcter, mira en quin lloc (entre 0 i 255) es troben els valors emmagatzemats a A i B i efectua la comparació numèrica de les posicions corresponents.

Així, si la variable o constant A està emmagatzemant el símbol '<', que en la taula ASCII és a la posició 60 i la variable o constant B està emmagatzemant el símbol 'Z', que en la taula ASCII és a la posició 90, resulta que l'ordinador avaluaria:

- com a certes les expressions: `A<B`, `A<=B`, `A!=B`
- com a falses les expressions: `A==B`, `A>B`, `A>=B`

Alerta! Les majúscules i minúscules i les vocals accentuades són caràcters diferents que ocupen posicions diferents a les taules de caràcters. Cal tenir-ho present. ‼

2.7. Expressions i ordre d'avaluació dels operadors

Sabem que existeixen els tipus de dades simples bàsics per a l'elaboració de programes i que es poden realitzar diferents operacions sobre aquestes dades. També sabem que la majoria d'operadors són binaris (exceptuant-ne la negació, que és unària). En els dissenys d'algorismes, però, moltes vegades apareixen expressions que combinen una gran quantitat d'operacions, com per exemple:

a) `( x + y ) * ( a - b )`
b) `(a + b ) div 2 < c` **`i`** `és_blanc`

Suposem que les dades de les anteriors expressions són dels tipus adequats a les operacions que s'hi utilitzen, és a dir:
a) `x`, `y`, `a`, `b` han de ser numèriques, ja que així té sentit considerar les expressions `x+y` i `a-b`, les quals també són dades numèriques, i per tant té sentit l'expressió proposada, la qual té, com a resultat, una dada numèrica.

b) `a` i `b` han de ser enteres, ja que així l'expressió `a+b` és entera i es pot efectuar l'operació `div` per 2 i obtenir un resultat enter que es vol comparar amb `c`, la qual ha de ser, òbviament, numèrica; el resultat de l'expressió `(a+b) div 2 < c` és, per tant, lògic i aquesta expressió està

seguida de `i és_blanc`, la qual cosa implica que `és_blanc` ha de ser una dada lògica; el resultat final de l'expressió és una dada lògica.

Què, com ho veieu? Segur que deveu pensar que el que hem fet no és altra cosa que l'avaluació d'una expressió, efectuada de manera similar a l'avaluació que estàveu acostumats a fer en matemàtiques: d'esquerra cap a dreta, però tenint en compte que algunes operacions tenen més prioritat que d'altres.

Una pregunta: quin és el resultat de l'expressió `3 + 4 * 5` ?
Esperem que la vostra resposta hagi estat `23`. Si no és així, teniu un greu problema de base matemàtica.

Si la vostra resposta ha estat `35` vol dir que no teniu clar l'ordre d'avaluació de les expressions matemàtiques. En aquest crèdit se suposa que aquest concepte el teniu plenament assolit. Per tant, si no és així, cal que cerqueu el suport necessari que us permeti entendre l'ordre d'avaluació en les expressions matemàtiques. (!!)

Quan tinguem expressions complexes, cal tenir molt clar quin serà l'ordre d'avaluació que seguirà l'ordinador. La prioritat en l'avaluació dels operadors en tota expressió és, ordenadament:

- parèntesis (començant pels més interns);
- potències (si el llenguatge els suporta -en pseudocodi, no en tenim-);
- productes i divisions (d'esquerra a dreta);
- sumes i restes (d'esquerra a dreta);
- concatenacions (pel tipus cadena que encara no hem vist);
- relacionals;
- negació;
- conjuncions i disjuncions (d'esquerra a dreta).

2.8. Tipus de dades simples en el llenguatge C

Vegem quines possibilitats facilita el llenguatge C per a la gestió dels tipus de dades simples.

2.8.1. Dades de tipus numèric

Detallarem els diferents tipus de dades numèriques que proporciona el llenguatge C amb les característiques que tenen i les operacions possibles.

1) Dades numèriques enteres i naturals

a) `short int` o, simplement, `short`

Dada per a emmagatzemar valors enters positius i negatius compresos entre –32768 (-2^{15}) i 32767 ($2^{15}-1$). Ocupa 2 octets. Pot ser precedida de la partícula `signed` per indicar que es tracta d'un tipus que pot "tenir signe", és a dir, pot emmagatzemar valors positius i negatius.

b) `int`

Dada per a emmagatzemar valors enters positius i negatius igual que el tipus anterior. La grandària en octets depèn de l'arquitectura de la màquina i de la versió del compilador de C. Si la màquina i el compilador són de 32 bits, estaríem davant una implementació del tipus `int` en 4 octets, però en arquitectures de 16 bits, s'implementa en 2 octets i llavors coincideix amb el tipus `short`. Por anar precedit per la partícula `signed`.

c) `long int` o, simplement, `long`

Dada per a emmagatzemar valors enters positius i negatius com les anteriors. El nombre d'octets depèn de l'arquitectura de la màquina i del compilador. Tant en les plataformes de 16 bits com en les 32 bits es correspondria amb 4 octets i els valors possibles estarien compresos entre –2147483648 (-2^{31}) i 2147483647 ($2^{31}-1$). En canvi, en plataformes de 64 bits, correspondria 8 octets i els valors possibles estarien compresos entre -2^{63} i $2^{63}-1$. Pot anar precedit de la partícula `signed`. Si necessitem la precisió de 64 bits i estem en plataforma de 32 bits, les darreres implementacions de C faciliten el tipus `long long int` (o `signed long long int`).

d) `unsigned short int` o, simplement, `unsigned short`

Dada per a emmagatzemar valors enters positius compresos entre 0 i 65535 ($2^{16}-1$). Ocupa 2 octets.

e) `unsigned int` o, simplement, `unsigned`

Dada per a emmagatzemar valors enters positius utilitzant 2 o 4 octets segons plataforma.

f) `unsigned long int` o, simplement, `unsigned long`

Dada per a emmagatzemar valors enters positius utilitzant 4 octets en plataformes de 16 i de 32 bits i amb valors possibles entre 0 i

4294967295 (2^{32}–1). En canvi, en plataformes de 64 bits utilitza 8 octets i els valors possibles estan compresos entre -2^{63} i 2^{63}–1. Si necessitem la precisió de 64 bits i estem en plataforma de 32 bits, les darreres implementacions de C faciliten el tipus `unsigned long long int`.

Cal utilitzar dades de tipus enter o natural quan els nostres càlculs hagin de treballar amb valors sense decimals. Escollirem short, int o long en les seves versions signed o unsigned, en funció de la grandària dels valors emmagatzemats.

Al llarg d'aquest llibre suposarem que la plataforma de màquina-compilador utilitzada és de 32 bits i, per tant, que el tipus int sigui de 4 octets.

El llenguatge C té una característica que no tenen la resta de llenguatges i que porta molts problemes als programadors inexperts i, fins i tot, als programadors experts procedents d'altres llenguatges.

Hem vist que tota dada numèrica entera té una capacitat finita. Què passa en temps d'execució quan el programa demana a l'ordinador que emmagatzemi en una variable entera un valor per al qual la variable no té capacitat suficient?

En molts llenguatges es produeix un error en temps d'execució que provoca la fi immediata del programa. En argot informàtic es diu que "el programa ha avortat". Quan això passa, el programa acostuma a informar d'un error d'overflow (valor per damunt de la capacitat permesa, tant en el vessant positiu com en el negatiu).

Doncs el nostre amic C no fa això. No provoca cap error i, per tant, no avorta l'execució del programa. Llavors, com actua? És difícil d'explicar amb paraules. Un exemple és, potser, el més convenient.

Llenguatge C: comportament cíclic en dades enteres

Suposem que i, j i k són variables de tipus `unsigned short`. Per tant ja sabem que poden emmagatzemar valors entre 0 i 65535. Suposem que en un moment determinat.

- i està emmagatzemant el valor 30000.
- j està emmagatzemant el valor 40000.

i volem (encara no sabem com programar-ho!) emmagatzemar dins k la suma dels valors que guarden i i j. És clar que el valor que esperem guardar dins k ha de ser 70000 i sabem també que no hi cap. Doncs "Mister C" no es queixa i hi guarda 4464. Com apareix aquest valor?

Imagineu-vos el conjunt de valors numèrics representables (0…65535) com un conjunt cíclic: així, d'igual manera que després del 60000 ve el 60001, succeeix que després del

65535 ve el 0 i després l'1, i així successivament. Per tant, com que 70000 excedeix el valor màxim, 65535, en 4465 unitats, tornant a començar pel valor mínim, que és 0, resulta que arribem a 4464, valor que és l'emmagatzemat dins k.

Aquest funcionament passa en tots els tipus de dades numèriques enteres.

2) Dades numèriques reals

a) `float`

Dada per a emmagatzemar valors reals de precisió simple (4 octets), és a dir, nombres reals que tenen un conjunt de dígits significatius relativament baix (concretament, 7 dígits significatius).

Els valors emmagatzemats estan compresos entre:

–3.402823E+38 i –1.175494E–38 per a nombres negatius.
1.175494E–38 i 3.402823E+38 per a nombres positius.

Exemples de notacions científiques

Com molt bé deveu saber, estem utilitzant notació científica per a expressar aquests valors; és a dir, el valor X que segueix la lletra E indica que cal multiplicar el valor anterior a la E per 10 elevat al valor X.

2.5E+5 indica 2.5 multiplicat per 10 elevat a 5, és a dir, 250000.
2.5E–4 indica 2.5 multiplicat per 10 elevat a –4, és a dir, 0.00025.
–2.5E+5 indica –2.5 multiplicat per 10 elevat a 5, és a dir, –250000.
–2.5E–4 indica –2.5 multiplicat per 10 elevat a –4, és a dir, –0.00025.

b) `double`

Dada per a emmagatzemar valors reals de precisió doble (8 octets), és a dir, nombres reals que tenen un conjunt de dígits significatius superior al tipus `float` (concretament, 16 dígits significatius).

Els valors emmagatzemats estan compresos entre:

–1.79769E+308 i –2.22507E–308 per a nombres negatius.
2.22507E–308 i 1.79769E+308 per a nombres positius.

c) `long double`

Dada per a emmagatzemar valors reals de precisió doble en format llarg (10 octets), és a dir, nombres reals que tenen un conjunt de dígits significatius superior, encara, al tipus `double` (concretament, 19 dígits significatius).

Els valors emmagatzemats estan compresos entre:

–1.189731E+4932 i –3.362103E–4932 per a nombres negatius.
3.362103E–4932 i 1.189731E+4932 per a nombres positius.

Cal utilitzar dades de tipus real quan els nostres càlculs hagin de treballar amb valors decimals. Escollirem float, double o long double en funció de la grandària dels valors emmagatzemats i de la quantitat de xifres significatives que calgui considerar.

3) Operadors numèrics

La correspondència en el llenguatge C amb els operadors numèrics que hem definit, de moment, en pseudocodi, és el que mostra la taula 6.

Taula 6. Equivalència operadors numèrics en pseudocodi i llenguatge C

Pseudocodi	+	–	*	/	div	mod
Llenguatge C	+	–	*	/	/	%

Ara bé, el llenguatge C disposa d'uns altres operadors per a facilitar la feina del programador. A continuació presentem els més utilitzats.

a) Operadors d'increment + + i decrement – –

Són operadors que poden acompanyar una variable numèrica per la dreta o per l'esquerra i la incrementen o decrementen en una unitat.

```
int n = 3; float r = 5.3;
n++; /* La variable n passa a contenir el valor 4; és equivalent escriure ++n */
f--; /* La variable f passa a contenir el valor 4.3; és equivalent escriure --f */
```

No sempre és equivalent aplicar aquests operadors per l'esquerra o per la dreta. Més endavant veurem quan hi ha diferència.

b) Operadors d'operació més assignació +=, -=, /=, *=, %=

Són operadors que consisteixen a sumar, restar, dividir, multiplicar o fer el mòdul del valor de la dreta sobre la variable de l'esquerra, assignant el resultat a aquesta variable.

```
int n = 5;
n += 3; /* És equivalent a fer n = n + 3; */
n -= 2; /* És equivalent a fer n = n - 2; */
n *= 4; /* És equivalent a fer n = n * 4; */
n /= 5; /* És equivalent a fer n = n / 5; */
n %= 2; /* És equivalent a fer n = n % 2; */
```

2.8.2. Dada de tipus caràcter

El llenguatge C aporta dos tipus de dada per al tractament del tipus caràcter.

Com que el conjunt de valors que cal representar consta de 256 elements, n'hi ha prou, en qualsevol cas, amb 1 octet.

1) `char`

C utilitza aquest tipus per a emmagatzemar un valor enter entre –128 i 127. Els valors 0 a 127 equivalen als 128 primers caràcters del codi ASCII. Els següents caràcters (des del que ocupa el lloc 128 fins al que ocupa el lloc 255) estan representats pels valors enters –128… –1.

2) `unsigned char`

C utilitza aquest tipus per a emmagatzemar un valor enter entre 0 i 255, el qual es correspon amb el caràcter que ocupa el lloc corresponent.

Per tant, en C es pot utilitzar els tipus caràcter per a tractar dades numèriques enteres, els valors de les quals siguin entre –128 i 127 per al tipus char i entre 0 i 255 per al tipus unsigned char.

2.8.3. Declaració de variables i constants

De manera semblant al pseudocodi, un programa en C tindrà la forma:

```
void main()
{
   /* declaració de constants i variables */
   /* instruccions del programa */
}
```

En C, a diferència del pseudocodi, podrem tenir les constants i les variables barrejades. Si ho preferiu, podeu posar primer les constants i després les variables, però no és obligatori.

La sintaxi que cal utilitzar per a declarar una dada en C és:

```
[const] <tipus> <nom_dada> [ = <valor> ] [,<nom_dada> [= <valor>] …];
```

Heu de començar a acostumar-vos a la notació que s'utilitza en informàtica:

- qualsevol element entre claudàtors [i] és optatiu;

- qualsevol element entre claus { i } és obligatori;

- qualsevol element en negreta indica paraula clau (reservada) que no admet cap altra forma d'escriptura; en alguns entorns s'utilitza la majúscula i, fins i tot, hi ha entorns en els quals les paraules clau no s'indiquen de cap manera especial; però en tal cas, les paraules no clau (escollibles pel programador) són les que van acompanyades d'alguna indicació;

- qualsevol element entre < i > indica paraula escollible pel programador;

- qualsevol llista d'elements separats per | indica diferents alternatives de les quals el programador n'haurà d'escollir una;

- qualsevol llista d'elements separats per ',' i acabada amb '…' indica una seqüència indefinida.

Comentem, però, el cas que ens ocupa:

- Utilitzarem `const` quan vulguem definir una constant; per defecte, estarem definint una variable.

- És optatiu acompanyar una dada d'un valor; en cas que ho fem, estem inicialitzant la dada. Evidentment, farem servir aquesta possibilitat per a inicialitzar les constants; en cas contrari, el valor que l'ordinador els assigni serà el que mantindran al llarg de l'execució del programa, ja que una constant no és modificable. Podem utilitzar també aquesta possibilitat per a donar un valor inicial a les variables; tant si ho fem com si no, posteriorment, disposarem de mecanismes per a modificar el valor que contenen.

- Els punts suspensius del final ens permeten utilitzar una mateixa declaració "[const] <tipus> …" per a declarar diverses constants-variables del mateix tipus.

Exemples:

```
const unsigned int edat = 25, ptes = 0;
float pes = 85.5, altura;
```

Cal tenir en compte els punts següents:

- En el primer exemple, hem declarat dues constants `unsigned int` de noms `edat` i `ptes` amb valors 25 i 0, respectivament.

- En el segon exemple, hem declarat dues variables `float` de noms `pes` i `altura` amb valor inicial 85.5 per a la primera; no podem dir res sobre el valor inicial que l'ordinador assignarà, en execució, a la variable `altura`, valor que pot canviar a cada execució del programa.

2.8.4. Especificació de valors constants

En aquest apartat farem referència a diferents sistemes de numeració utilitzats en informàtica (decimal –base 10–, octal –base 8– i hexadecimal –base 16–).

Com podem indicar en un programa que un valor constant és d'un determinat tipus? És a dir, quan en un programa s'escrigui la constant 3, com pot el programador indicar al compilador si aquest 3 és real, enter, etc.?

Una constant en llenguatge C pot ser un enter, un real o un caràcter.

1) Constants enteres

La sintaxi per a especificar una constant entera és:

```
{[+] | –} [{0|Ox|0X}]<valor>[{L|U|UL}]
```

és a dir:

- Si és positiva pot portar o no el símbol +, però haurà de portar obligatòriament el símbol – si és negativa.

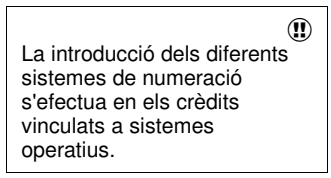
- El valor pot tenir per prefix els símbols 0 (zero), 0x (zero més x), 0X (zero més X) o no tenir-ne.

– Sense prefix: indica una constant en base 10, la qual pot contenir un o més dígits del 0 al 9, el primer dels quals ha de ser diferent de zero.

– Amb prefix 0: indica una constant en base 8, la qual pot tenir un o més dígits del 0 al 7.

– Amb prefix Ox o OX: indica una constant en base 16, la qual pot tenir un o més caràcters del 0 al 9 i de la A a la F, en majúscules o minúscules.

- El valor pot tenir per sufix els símbols U, L, UL o no tenir-ne.



– Sense sufix : si és en base 10, el seu tipus és el primer dels tipus `int`, `long` i `unsigned long` en el qual pugui ser representat. Si és en base 8, el seu tipus és el primer dels tipus `int`, `unsigned int`, `long` i `unsigned long` en el qual pugui ser representat. Si és en base 16, el seu tipus és el primer dels tipus `int` o `unsigned int` en el qual pugui ser representat.

– Amb sufix L: el seu tipus és `long` quan el valor pot ser representat per aquest tipus (és dins l'interval de valors corresponent); en cas contrari, és considerat `unsigned long`.

– Amb sufix U: el seu tipus és `unsigned int` quan el valor pot ser representat per aquest tipus; en cas contrari és considerat `unsigned long`.

– Amb sufix UL: el seu tipus és `unsigned long`.

Exemples de representació de constants enteres en el llenguatge C

`2500U` Constant entera en base 10 representada com a `unsigned int`

`10000` Constant entera en base 10 representada com a `int`

`50000` Constant entera en base 10 representada com a `int` en plataforma de 32 bits, però representada com a `long` en plataforma de 16 bits ja que 50000 no és representable com a `int` en plataforma de 16 bits

`0350000` Constant entera en base 8. El seu valor en base 10 és 118784 i, per tant, en plataforma de 32 bits és representada com a `int` però en plataforma de 16 bits seria representada com a `long`.

`–035000` Constant entera en base 8 representada com a `int` ja que el seu valor en base 10 és –14848, que pot ser representat, en primer lloc, com a int

`0x35AFF` Constant entera en base 16. El seu valor en base 10 és 219903 i, per tant, en plataforma de 32 bits és representada com a `int` però en plataforma de 16 bits seria representada com a `long`.

2) Constants reals

La sintaxi per a especificar una constant real és:

```
{[+] | –}<part_entera>.<part_fraccionària>[{e|E}{[+] | –}<exponent>][f|F|l|L]
```

en què:

- Si és positiva pot portar o no el símbol +, però haurà de portar obligatòriament el símbol – si és negativa.
- <exponent> representa zero o més dígits del 0 al 9.
- E o e és el símbol d'exponent de la base 10, que pot ser positiu o negatiu (E–3 indica 10 elevat a –3).

- Pot portar un sufix f o F o l o L, o no portar-ne.

– Sense sufix: el seu tipus és `double`.
– Amb sufix f o F: el seu tipus és `float`.
– Amb sufix l o L: el seu tipus és `long double`.

Exemples de representació de constants reals en el llenguatge C

`-15.45F`	Constant de tipus `float` amb valor –15.45
`12.30E-4`	Constant de tipus `double` amb valor 0.00123
`.001e5L`	Constant de tipus `long double` amb valor 100

3) Constants caràcter

Són sempre de tipus `char` i consisteixen en un únic caràcter entre cometes simples. A vegades, però, es pot fer referència al caràcter per mitjà del seu codi ASCII en hexadecimal. En tal cas, cal posar el valor precedit dels caràcters \x i també entre cometes simples. (!!)

Les seqüències d'escapament (caràcters de les posicions 0 a 31, en el codi ASCII) són considerades com un únic caràcter i el llenguatge C acostuma a facilitar-ne tenir alguna nomenclatura especial.

Exemples de representació de constants caràcter en el llenguatge C

`'A'`	Constant `char` corresponent a la lletra majúscula A
`'\x41'`	Constant `char` corresponent a la lletra majúscula A ja que 41 és el codi ASCII
`'\x10'`	Constant `char` corresponent a la seqüència d'escapament LF (salt de línia)
`'\n'`	Constant `char` corresponent a la seqüència d'escapament LF (salt de línia)

2.8.5. Tractament del tipus lògic

El llenguatge C no disposa d'un tipus específic per representar les dades lògiques. Com s'ho manega, doncs? C utilitza les dades numèriques per a tal finalitat.

> En el llenguatge C, tota dada numèrica amb valor igual a zero és considerada falsa a efectes lògics, i tota dada numèrica amb valor diferent de zero és considerada certa a efectes lògics.

És a dir, si tenim:

```
int n = 10, p = 0;
char c = 'A';
float f = 0;
```

el llenguatge C, interpretant-les com a variables lògiques, tindria `n` i `c` com a certes i `p` i `f` com a falses.

Els programadors acostumen a utilitzar variables enteres per al tractament dels valors lògics. (!!)

Operadors lògics

La correspondència en el llenguatge C amb els operadors lògics que hem definit en pseudocodi és el que mostra la taula 7. Observar que el llenguatge C no facilita un operador lògic per a la disjunció exclusiva.

Taula 7. Equivalència operadors lògics en pseudocodi i llenguatge C

Pseudocodi	i	o	xor	no
Llenguatge C	&&	||		!

Operadors relacionals

La correspondència en el llenguatge C amb els operadors relacionals que hem definit en pseudocodi és:

Pseudocodi	<	>	≤	≥	==	!=
Llenguatge C	<	>	<=	>=	==	!=

2.8.6. Conversió de tipus

Quan els operands que intervenen en una operació són de tipus diferents, abans d'efectuar l'operació especificada es converteixen a un tipus comú, d'acord amb les regles que s'exposen a continuació:

- Els operands que intervenen en una determinada operació són convertits al tipus de l'operand de precisió més alta.

- En C, les constants reals són considerades de tipus `double` per defecte.

- Una expressió lògica dóna 1 com a resultat `cert` i 0 com a resultat `fals`.

- En una assignació, el valor de la part dreta és convertit al tipus de la variable de l'esquerra, d'acord amb les regles següents:

– Els caràcters es converteixen a enters.

– Els enters es converteixen a caràcters, preservant els bits de menys pes, és a dir, desaprofitant els bits de més pes en excés.

– Els reals són convertits a enters, truncant la part fraccionària.

– Un `double` passa a `float`, arrodonint i perdent precisió si el valor `double` no pot ser representat exactament com a `float`.

En C està permesa una conversió explícita del tipus d'una expressió mitjançant una construcció denominada `cast`, que té la forma que segueix a continuació:

```
(<nom_de_tipus>) expressió
```

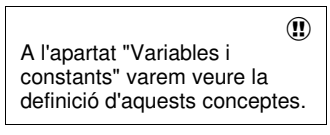
L'expressió és convertida al tipus especificat si la conversió és permesa; en cas contrari, es produirà un error. La utilització apropiada de construccions cast garanteix una avaluació consistent, però sempre que es pugui és millor evitar-la, ja que suprimeix la verificació de tipus proporcionada pel compilador i, per tant, pot conduir a resultats inesperats.

```
float f = 5.2;
(double) f ;      /* Expressió que considera el valor 5.2 de f com a double */
```

2.8.7. Constants simbòliques

El llenguatge C proveeix un altre mètode de declaració de constants diferent de la utilització del mecanisme `const`. En realitat no són constants en el sentit estricte de la paraula. Són les anomenades constants simbòliques, utilitzades com a constants per molts programadors.

Sabem que les dades ocupen memòria i que les classifiquem en variables i constants en funció de si el seu valor es pot modificar o no al llarg de l'execució del programa. Per tant, una constant és una dada que ocupa memòria i que conté un valor des del seu inici. Suposem, per exemple:




```
const edat = 25;
```

Tal com veiem a la figura 2, 25 està ocupant part de la memòria mentre s'executa el programa.

Declarar una constant simbòlica és una manera de comunicar al programa compilador un nom que porta associat un valor, de manera que, en el procés de traducció, el programa compilador substituirà totes les aparicions del nom pel valor en qüestió.

Figura 2. Exemple de dada constant ocupant part de la memòria RAM

La sintaxi que s'utilitza és:

```
#define <nom_constant_simbòlica> <valor>
```

Exemples de declaració de constants simbòliques:

```
#define edat 25
#define ptes 0
```

Utilitzant constants simbòliques, l'ordinador no assigna cap zona de memòria en temps d'execució. És a dir, en execució, el processador no es trobarà cap instrucció en què es faci referència a una constant de nom `edat` o `pes`, sinó que trobarà directament els valors 25 i 0, respectivament.

En canvi, quan es treballa amb constants en el sentit informàtic de la paraula, en execució, el processador troba els noms `edat` i `pes` en les instruccions que correspongui i mitjançant el nom accedeix a la memòria per tal de cercar el valor.

I ara segur que us feu la pregunta del milió: quan codifiquem amb el llenguatge C, què és millor utilitzar: constants tipus `const` o constants tipus `#define`? Ja veureu que, de moment, farem servir sempre `#define`, és a dir, no aplicarem el concepte informàtic de constant. El motiu és clar: si no ocupen memòria, per què caldria utilitzar un altre mètode que sí que n'ocupa?

La modificació de les constants

Fins el moment hem introduït molts conceptes sobre la utilització de la memòria en temps d'execució per mitjà de constants i variables. I hem convingut que una constant és una dada immodificable durant l'execució del programa.

Aquesta darrera afirmació és certa a mitges. Una constant, durant l'execució del programa, ocupa memòria. El processador té accés absolut a la memòria RAM. Per tant, assegurar que NO ÉS MODIFICABLE és força arriscat, ja que si el processador hi té accés absolut, podria, si volgués, modificar-la. O no?

Us heu d'imaginar la memòria com una habitació que conté les constants i les variables i que té, com a accés, dues portes: una per a les constants i una altra per a les variables. Imagineu-vos el programa que està en execució. Quan ha d'accedir a una constant, cosa que fa per mitjà del seu nom, entra a la memòria per la porta de les constants; per tant, només pot veure (llegir) el valor emmagatzemat. Per al cas de les variables, entra a la memòria per la porta de les variables; per tant, hi pot llegir i escriure.

En realitat hi ha un tercer accés. Com molt bé sabeu, la memòria està dividida en octets, els quals estan numerats. És a dir, cada zona de la memòria té una adreça i el processador pot anar a cada adreça i fer-hi el que li sembli oportú. Per tant, la realitat és que tota dada emmagatzemada a la memòria és accessible i pot ser modificada si es coneix l'adreça on el processador l'ha deixat. I molts llenguatges de programació –entre els quals hi ha el nostre amic C– faciliten al programador la possibilitat d'accedir a la memòria mitjançant adreces. Però aquest és un tema que s'escapa dels objectius actuals.

2.8.8. Introducció al precompilador C

En aquest apartat estudiarem el precompilador C i aprendrem el concepte de directriu.

El precompilador C és un processador de text que manipula el text d'un arxiu font (arxiu d'extensió C) com a part d'una primera fase de traducció i abans que la fase de compilació pròpiament dita comenci.

El precompilador avalua tot l'arxiu font, però només en manipula les parts conegudes com a directrius pel precompilador.

Una directriu és una instrucció per al precompilador de C.

Les directrius són utilitzades per a fer programes fàcils de canviar i fàcils de compilar en diferents entorns d'execució. Les directrius indiquen al precompilador quines accions específiques ha d'executar. Per exemple:

- substituir elements en el text (utilitzant `#define`);
- inserir contingut d'altres arxius (utilitzant `#include`);
- suprimir la compilació de parts de l'arxiu font (amb `#if...#endif`).

És a dir, la declaració d'una constant simbòlica no és altra cosa que la utilització d'una directriu de precompilació, de manera que, abans d'iniciar la veritable fase de traducció del programa font, actua el

precompilador i substitueix totes les aparicions de la constant simbòlica pel valor assignat.

Com que les directrius de precompilació s'avaluen en una fase inicial, hi ha el costum entre els programadors en C de declarar-les al començament dels arxius font, abans i tot de la línia on apareix la paraula reservada `main`.

En aquest moment no entrem en profunditat en la utilització de les directrius de precompilació. A mesura que les necessitem les anirem introduint.

2.9. Tipus de dades simples definibles

En aquests moments coneixem els tipus de dades simples (per a emmagatzemar un únic valor) que ens proporcionen la majoria de llenguatges. Alguns llenguatges permeten al programador la definició d'altres tipus de dades per a facilitar el disseny d'algorismes.

Els tipus de dades simples definibles es divideixen en dos grans apartats: enumerades i subrangs o subtipus.

2.9.1. Tipus de dades enumerades

Una enumeració és un procés pel qual s'estableix una correspondència, d'un a molts, entre una variable entera i els únics valors que aquesta pot prendre.

Els valors de la variable, que es defineixen com a possibles dins l'enumeració, estan associats a identificadors que, si no s'inicialitzen, prenen valors correlatius.

Enumeracions en pseudocodi

La sintaxi que farem servir és:

```
enum <nom> = (<identificador> [,<identificador> ...]) ;
```

En pseudocodi suposarem que els identificadors no s'inicialitzen. Els elements de l'enumeració tenen un ordre que és precisament l'ordre en què s'han escrit els valors. Veureu, més endavant, que en llenguatge C no és exactament així.

Funcionalitat dels llenguatges

Per a treballar amb un llenguatge que facilita l'enumeració cal cercar informació corresponent a la seva funcionalitat. No tots els llenguatges en proporcionen la mateixa. Ara bé, sí que en tots és semblant.

Un exemple val més que mil paraules, oi? Suposem que en el nostre programa hem de treballar amb els dies de la setmana. Potser estarà bé disposar d'un tipus de dada que ens permeti treballar amb els dies, passant al següent, a l'anterior, etc. Som-hi!

```
enum dies = (dilluns, dimarts, dimecres, dijous, divendres, dissabte, diumenge) ;
```

Amb aquesta declaració acabem de definir un nou tipus de dada simple que podem fer servir en el nostre programa.

Una vegada tenim declarada l'enumeració, podrem definir variables d'aquest tipus i operar-hi juntament amb els operadors que veurem a continuació.

Així, doncs, podem declarar:

```
var
   dia : dies;
fivar
```

i podrem fer assignacions del tipus:

```
dia = dilluns;
```

Veient l'exemple, no us pregunteu on es declara l'enumeració? Bé, hem dit abans que l'enumeració és un nou tipus i ja hem comprovat que podem declarar variables d'aquest nou tipus. Sembla lògic pensar que és necessari haver declarat l'enumeració abans de declarar les corresponents variables. En efecte, però encara hi ha més coses que cal tenir en compte.

El concepte d'enumeració ha estat el primer que ens ha permès declarar tipus nous en els nostres programes. Aquesta pràctica és fonamental per a poder dissenyar algorismes entenedors i eficients i, a mesura que avancem en aquest crèdit, anirem aportant maneres de declarar nous tipus.

Totes tenen un denominador comú: la declaració de tipus. En pseudocodi és obligatori declarar els tipus en una zona especial per a aquesta

finalitat (de manera semblant a la declaració de constants i de variables). Tenim, doncs, que l'estructura d'un programa passa a ser la següent:

```
programa <nom programa> és
   const <nom_constant_1> = <valor_1>;
         <nom_constant_2> = <valor_2>;
         ...
   ficons
   tipus <declaració_tipus1>;
         <declaració_tipus2>;
         ...
   fitipus
   var <nom_variable_1> : <tipus_variable_1>;
       <nom_variable_2> : <tipus_variable_2>;
       ...
   fivar
   <instruccions del programa>
fiprograma
```

Així, en l'exemple anterior, hauríem escrit:

```
tipus
   enum dies = (dilluns, dimarts, dimecres, dijous, divendres, dissabte, diumenge) ;
fitipus
var
   dia : dies;
fivar
```

Operadors sobre els tipus de dades enumerats

En realitat són funcions, terme pendent de tractar en profunditat, i que implica que retornen un resultat. Donada una variable x corresponent a un tipus de dada enumerat, tenim les operacions (funcions) següents:

- `suc(x)`, que retorna el valor de la dada enumerada immediatament superior a x; si x és la més gran, retorna la mateixa x;
- `pre(x)`, que retorna el valor de la dada enumerada immediatament inferior a x; si x és la més petita, retorna la mateixa x;
- `ord(x)`, que retorna un nombre natural corresponent al número d'ordre de x dins l'enumeració (suposarem que comencen a comptar des de zero).

suc(x)

El nom prové de successor.

pre(x)

El nom prové de predecessor.

ord(x)

El nom prové d'ordinal.

Aquests tipus de dades permeten fer servir els operadors relacionals.

Enumeracions en llenguatge C

El llenguatge C permet, també, els tipus enumerats, però amb diferències substancials respecte del que hem definit en pseudocodi.

La sintaxi que hem d'utilitzar és:

```
enum <nom>
{
   <identificador>[=<valor_enter>]
   [,<identificador>[=<valor_enter>] ...]
};
```

Sobre això, cal comentar el següent:

- A diferència del pseudocodi, els diferents identificadors es poden inicialitzar amb valors enters, que no tenen per què ser correlatius.
- Qualsevol identificador es pot inicialitzar amb un valor enter resultat d'una expressió. Els identificadors successius prendran valors correlatius a partir d'aquest si no és que també tinguin assignat un valor.
- Per defecte, el primer identificador s'inicialitza amb el valor zero.
- Dos o més identificadors poden tenir el mateix valor.
- Un mateix identificador no pot aparèixer en diferents tipus de dades.
- No és possible llegir o escriure directament un valor d'un tipus enumerat.

Considerem alguns exemples.

```
enum dies {dilluns, dimarts, dimecres, dijous, divendres, dissabte, diumenge};
```

En aquest cas, `dies` és un nou tipus que té set possibles valors naturals (0...6) identificats pels noms `dilluns`, `dimarts`, ... `diumenge`:

```
enum dies {dilluns=-1, dimarts=1, dimecres=-2, dijous=2, divendres=0, dissabte=1,
           diumenge=2};
```

En aquest cas, `dies` és un nou tipus que té els valors naturals següents:

-1 identificat per `dilluns`
1 identificat per `dimarts` i `dissabte`
-2 identificat per `dimecres`

2 identificat per `dijous` i `diumenge`
0 identificat per `divendres`

Com podeu observar, el tipus enumerat del llenguatge C és més potent que el tipus enumerat del pseudocodi. Ara bé, donada una variable en C declarada a partir d'un tipus enumerat, C la considera de tipus `int` i, per tant, aquesta variable pot rebre qualsevol valor de tipus `int` per mitjà dels conductes de modificació de valor d'una variable (instrucció d'assignació o instrucció de lectura). És a dir, C dóna total llibertat al programador per a tractar la variable com un `int`.

Límits del llenguatge C

El llenguatge C és molt potent i aquesta potència es troba, precisament, en el fet que aquest llenguatge té per màxima "el programador ja sap el que es fa; jo l'obeeixo". Per tant, alerta! És responsabilitat del programador en C controlar els valors que prenen les variables d'un tipus enumerat.

Als Estats Units ens interessaria identificar els dies de la setmana com a 0 per al diumenge, 1 per al dilluns, ..., 6 per al dissabte.

Un darrer comentari respecte de l'últim exemple: no pretengueu cercar cap lògica en aquesta definició. La majoria de les vegades ens interessarà identificar els dies de la setmana des d'1 –dilluns– fins a 7 –diumenge– (sobretot si som a Catalunya).

Com es defineix en C una variable d'un tipus `enum`? Tenim diverses formes de fer-ho. En primer lloc, vegem algunes maneres de declarar el nou tipus (no totes):

- Definició del tipus dins la declaració de variables abans de declarar cap variable del tipus en qüestió. Posteriorment es poden declarar variables del tipus.

```
enum dies {dilluns, dimarts, dimecres, dijous, divendres, dissabte, diumenge};
/* tipus */
enum dies dia1, dia2, dia3; /* variables */
```

- Definició del tipus dins la declaració de variables abans de declarar cap variable del tipus en qüestió, aprofitant la pròpia declaració de tipus per declarar una o més variables. Posteriorment es poden declarar més variables del tipus.

```
enum dies {dilluns, dimarts, dimecres, dijous, divendres, dissabte, diumenge} dia1,
dia2; /* tipus i dues variables */
enum dies dia3; /* variable */
```

Per acabar, en llenguatge C no hi ha operadors específics per al tractament de variables de tipus `enum`. Cal fer servir els operadors del tipus `int`.

2.9.2. Subtipus de dades en pseudocodi

Alguns llenguatges incorporen la possibilitat de delimitar el conjunt de valors possibles per als tipus `enter`, `natural` i `caràcter` (tipus estàndards) i per als tipus `enumerat` (definit pel programador), definint un nou tipus que és considerat `subtipus` del tipus del qual els seus valors en són subconjunt.

Les operacions permeses en els `subtipus` són les mateixes que les dels tipus dels quals deriven.

En llenguatge C no hi ha la possibilitat de declarar `subtipus`; en canvi, ja sabem que el tipus `enum` del llenguatge C és molt més potent que el tipus `enumerat` vist en pseudocodi.

Per tant, només farem referència a `subtipus` en pseudocodi. La sintaxi és la següent:

```
tipus
   <nom_tipus> = <llista_valors_possibles> | <interval_valors_possibles>;
...
fitipus
```

Exemples de definició de subtipus:

```
tipus
   sexe = 'H', 'D';
   trimestres = 1,2,3,4; /* o també 1..4 */
   notes = 0..10;
   enum dies = (dilluns, dimarts, dimecres, dijous, divendres, dissabte, diumenge);
   diesfeiners = dilluns...divendres;
   diesfestius = dissabte...diumenge;
fitipus
```

2.9.3. Redefinició de tipus de dades en C

El llenguatge C permet redefinir el nom dels tipus de dades estàndards que hi ha i, fins i tot, els tipus de dades creats pel programador, utilitzant la paraula reservada `typedef`, abreviatura de definició de dada però, realment, redefinició de nom de tipus de dada.

La sintaxi d'utilització d'aquesta instrucció és la següent:

```
typedef <nom_actual_tipus> <nom_nou_tipus>;
```

Com a exemple, podríem efectuar les següents redefinicions per apropar més el llenguatge C al pseudocodi:

```
typedef unsigned int natural4;
typedef unsigned long natural8;
typedef int enter4;
typedef long enter8;
typedef float real4;
typedef double real8;
typedef char caracter;
```

Posteriorment podrem definir variables utilitzant els noms nous:

```
natural4 edat;  /* unsigned int */
natural8 ptes;  /* unsigned lont */
real4 euros;    /* float */
```

3. Instruccions seqüencials bàsiques

Una vegada coneixem els tipus de dades simples que es fan servir en pseudocodi i la seva implementació en el llenguatge C així com les operacions que podem efectuar amb elles, ens manca saber com hem de lligar les diferents operacions per tal de plasmar en l'algorisme la seqüència de passos per a aconseguir la solució desitjada.

> Una estructura seqüencial és un conjunt d'instruccions que s'executen l'una a continuació de l'altra.

Una vegada executada la primera instrucció s'executen totes les altres en l'ordre en què s'han descrit.

Quan es dissenya una estructura seqüencial en pseudocodi s'acostuma a escriure una instrucció, acabada en ';', a cada línia. Podem, però, escriure diverses instruccions per línia. La bona pràctica aconsella fer servir el mètode d'una instrucció per línia, ja que s'assoleixen algorismes més clars per a futurs seguiments.

Per tant, un algorisme en pseudocodi tindrà la forma següent:

```
programa <nom programa> és
   [
      const ...
      ficonst
   ]
   [
      var ...
      fivar
   ]
      instrucció 1;
      instrucció 2;
      instrucció 3;
      ...
      instrucció N;
fiprograma
```

En el llenguatge C se segueix exactament la mateixa sintaxi descrita pel pseudocodi: instrucció darrera instrucció acabada cadascuna amb ";".

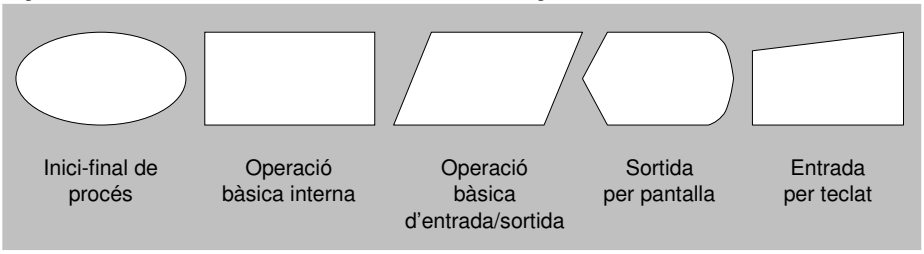
3.1. Ordinogrames

Els ordinogrames o diagrames de flux del procés representen la seqüència lògica de les operacions que cal fer per a dur a terme el procés.

Els símbols més usats habitualment en els ordinogrames, recollits a la figura 3, són: inici-final del procés, operació bàsica interna, operació bàsica d'entrada-sortida de perifèrics, preses de decisió i subprocés.

A la unitat didàctica "Estructures de control" veurem els símbols utilitzats en les preses de decisió.

Figura 3. Símbols més usats habitualment en els ordinogrames.

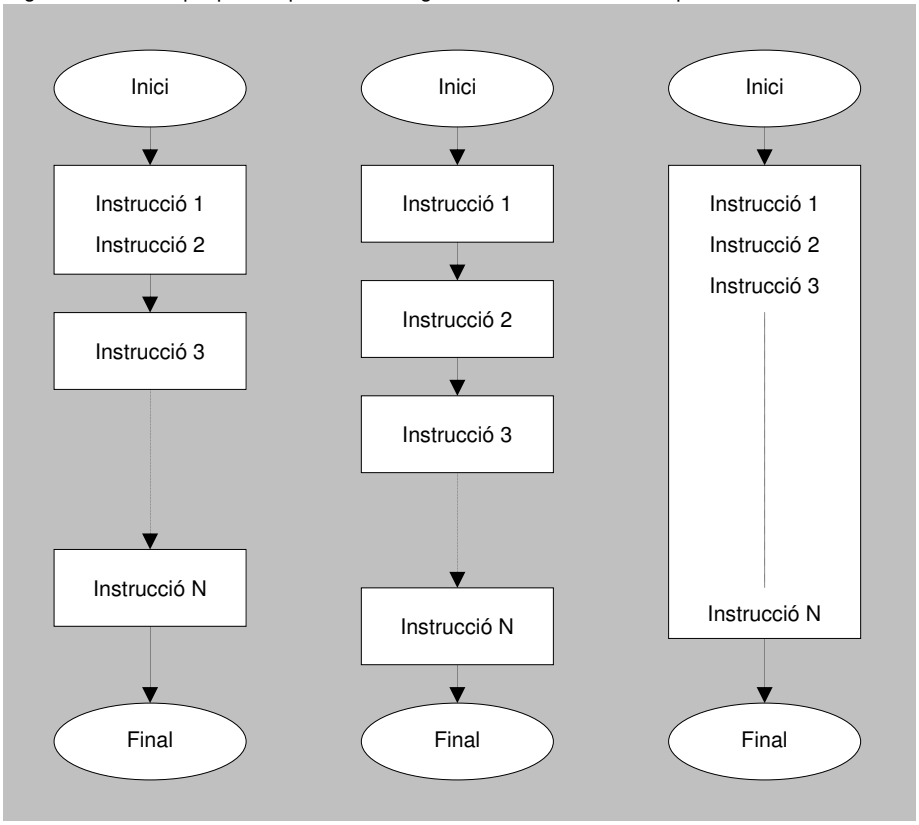


Perifèric

Anomenem perifèric tot objecte que permet la comunicació del processador amb l'exterior (pantalla, teclat, mòdem, impressora, altaveu, etc.).

La figura 4 mostra les formes que pot adoptar un ordinograma d'una estructura seqüencial.

Figura 4. Formes que pot adoptar un ordinograma d'una estructura seqüencial



3.2. Instrucció d'assignació

La primera instrucció que hem d'estudiar és la que assigna el valor a una variable. De fet, ja l'hem fet servir, de manera amagada, quan hem donat el valor inicial a l'hora de declarar les variables.

La instrucció d'assignació és la que permet donar valor a una variable.

La sintaxi utilitzada en pseudocodi és, principalment, la següent:

```
<nom_variable> ← <expressió> /* Notació pròpiament de pseudo-codi */
<nom_variable> := <expressió> /* Notació tipus Pascal */
<nom_variable> = <expressió> /* Notació tipus C –la que utilitzarem– */
```

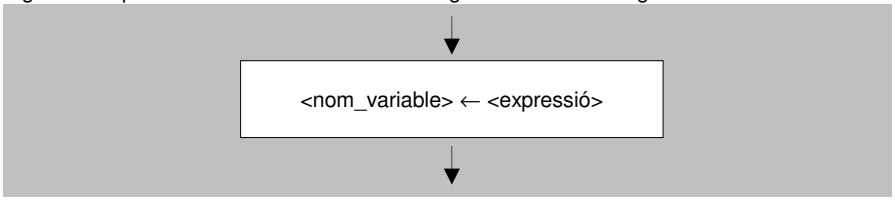
Una expressió és una combinació de variables, constants i operadors que, després d'haver estat avaluada segons les lleis de la matemàtica i la lògica, retornen un resultat.

En concret, una expressió pot estar constituïda per un únic valor.

Cal suposar que els tipus de la variable i del resultat de l'expressió són equivalents o compatibles. És a dir, no és possible assignar un resultat lògic a una variable de tipus numèric o caràcter. Ara bé, sí que es pot assignar un resultat numèric enter a una variable de tipus real, per exemple. (!!)

Per a representar una assignació en un ordinograma farem servir un símbol d'operació bàsica interna, com es mostra en la figura 5.

Figura 5. Representació d'una instrucció d'assignació en un ordinograma.



En llenguatge C se segueix exactament la sintaxi descrita pel pseudocodi, que consisteix en la utilització del símbol '**=**'. Per a no confondre'l amb l'operador relacional d'igualtat, es representa, com molt bé deveu recordar, amb un doble símbol d'igualtat '**==**'.

3.3. Instruccions d'entrada

La immensa majoria de programes han de gestionar dades que provenen de l'exterior (des del punt de vista de l'ordinador), les quals són introduïdes mitjançant els perifèrics d'entrada, a petició del programa que les ha de gestionar.

> Les instruccions d'entrada són les que permeten al programa rebre informació de l'exterior.

De manera semblant al funcionament de l'ésser humà quan rep informació, el qual l'emmagatzema inicialment en el cervell per tal de tractar-la posteriorment, els programes, quan reben informació, l'han d'emmagatzemar a la seva memòria i, per tant, la utilització d'instruccions d'entrada implica la corresponent utilització de variables.

A causa del fet anterior, i perquè hi ha diferents canals per a introduir informació a l'ordinador (el teclat, els suports magnètics, el mòdem, el ratolí, etc.), els llenguatges de programació necessiten instruccions d'entrada especialitzades (per tipus de dada i per canal d'entrada).

3.3.1. Instruccions d'entrada en pseudocodi i en ordinogrames

En pseudocodi, de moment, no ens complicarem la vida i considerarem que l'entrada es produeix, únicament, per teclat.

La instrucció d'entrada en pseudocodi és:

```
llegir ( <nom_variable> [, <nom_variable> ...] );
```

en la qual s'observa que en una instrucció `llegir` el programa pot rebre un o més valors introduïts des de l'exterior.

És obligatori que les variables hagin estat declarades prèviament i han de ser del tipus de dada que s'ha de recollir. En pseudocodi suposarem que la dada que arribi de l'exterior sempre serà del tipus adequat a la variable que l'ha de rebre. Quan codifica els algorismes en un llenguatge informàtic concret, el programador ha d'adoptar les mesures pertinents per a controlar la possibilitat que la dada rebuda sigui del tipus adequat, control que depèn del llenguatge concret.

Control de les instruccions d'entrada

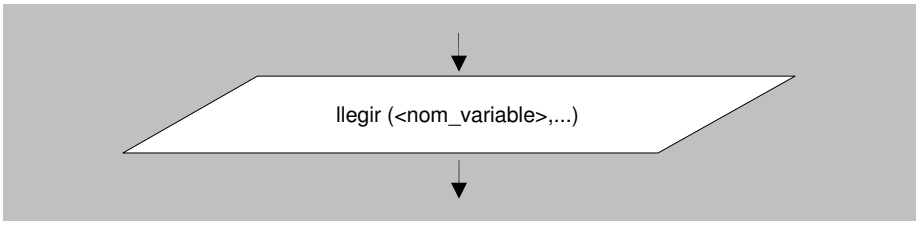
Hi ha llenguatges de programació que tenen un fort control en les instruccions d'entrada, de manera que si, per exemple, l'entrada s'efectua per teclat i l'usuari tecleja una dada de tipus erroni, el programa es queda aturat fins que l'usuari introdueix una dada del tipus esperat. Aquest és el comportament normal quan, per exemple, s'està demanant una dada numèrica i l'usuari, per equivocació, introdueix caràcters.

Estem parlant de control del tipus de dada en l'entrada. Això no s'ha de confondre amb el domini de valors que es poden introduir. Per exemple, si l'usuari ha d'entrar un valor numèric natural entre 0 i 9 i introdueix un 15, aquí no hi ha error de tipus de dada; estem davant un error del valor introduït. Els errors de valors sempre els ha de controlar el programador; hi ha llenguatges que aporten ajudes per a aquest tipus de control.

L'error a què ens referíem a l'inici és error de tipus de dada. Hi ha llenguatges que ja ho controlen per si mateixos. D'altres no ho fan i, en aquests casos, és responsabilitat del programador. Dins el grup de llenguatges que no ho controlen podem distingir dos subgrups: llenguatges que, en cas de produir-se un error no controlat pel programador, avorten l'execució del programa (llenguatges actius o agressius) i llenguatges que, en el mateix cas, continuen l'execució sense haver llegit la dada, de manera que la variable de recepció no ha modificat el seu valor (llenguatges passius). El llenguatge C pertany al grup de llenguatges que no ho controlen i que deixen prosseguir l'execució del programa.

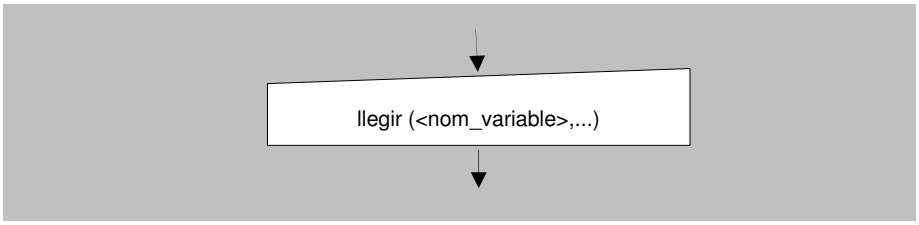
Per representar una instrucció d'entrada en un ordinograma farem servir un símbol d'operació bàsica d'entrada, com es mostra a la figura 6.

Figura 6. Representació d'una instrucció d'entrada en un ordinograma.



Si l'entrada es produeix per teclat podem utilitzar el símbol corresponent i tindríem la representació que es mostra a la figura 7.

Figura 7. Representació d'una instrucció d'entrada per teclat en un ordinograma.



Per últim, comentar que la instrucció llegir no es pot utilitzar per a efectuar l'entrada de qualsevol tipus de dada. Així, per exemple, no es pot utilitzar per a introduir valors lògics ni valors de tipus enumerats. ‼

3.3.2. Instruccions d'entrada en llenguatge C

En el llenguatge C tenim moltes instruccions corresponents al `llegir` del pseudocodi. Ens centrarem, de moment, en una única instrucció: la funció `scanf`.

La funció (penseu instrucció) `scanf` llegeix dades de l'entrada estàndard (anomenada `stdin` en llenguatge C i que acostuma a ser el teclat), les interpreta d'acord amb el format indicat i les emmagatzema en els llocs indicats.

Canals d'entrada i de sortida

Com ja deveu saber, l'ordinador pot tenir diversos canals d'entrada i diversos canals de sortida, dels quals n'hi ha un que és el canal d'entrada estàndard (normalment el teclat) i un que és el canal de sortida estàndard (normalment la pantalla), tot i que els canals estàndards es poden redireccionar, mitjançant el sistema operatiu, cap a altres canals.

En llenguatge C, el canal d'entrada estàndard s'anomena `stdin` i el canal de sortida estàndard, `stdout`.

La funció `scanf` té moltes possibilitats. Però per entendre-les hem d'esperar a tenir més coneixements del llenguatge C. De moment, farem servir la sintaxi d'aquesta funció imprescindible per a poder llegir dades per teclat. No intenteu, en aquest moment, sense ser experts en programació i en el llenguatge C, cercar informació d'aquesta funció enlloc. Us embolicaríeu de manera innecessària.

La sintaxi és:

```
scanf ("<format de lectura>", &<nom_variable>[, &<nom_variable>, ...] );
```

en la qual hi ha d'haver, com a mínim, una variable, la qual ha d'estar prèviament declarada.

Sintaxi de la instrucció `scanf`

El programa traductor de C necessita, en temps de compilació, informació sobre la sintaxi de la instrucció `scanf`. Aquesta informació la cerca, de manera automàtica, en certs directoris definits en l'entorn de programació on es desenvolupa el programa. A vegades, però, aquests directoris no estan definits o no contenen la informació cercada. Llavors cal informar-ne explícitament al compilador, la qual cosa es fa mitjançant la directriu de precompilació `#include`.

La directriu `#include` diu al precompilador que inclogui el fitxer especificat en el programa font, en el lloc on apareix la directriu. Hi ha dues maneres d'utilitzar-la:

- La sintaxi

```
#include "arxiu"
```

que provoca que l'arxiu es busqui en primer lloc en el directori actual de treball i posteriorment en els directoris estàndards definits en l'entorn de programació.

- La sintaxi

```
#include <arxiu>
```

que provoca que l'arxiu es busqui únicament en els directoris estàndards definits.

En el cas que ens ocupa, la informació necessària per a la instrucció `scanf` es troba a l'arxiu `stdio.h` (*standard input output*) que acostuma a estar en el directori `include` de l'entorn de programació.

Aquestes directives `#include` se solen situar al començament del programa font.

En cas de fer servir el llenguatge C++, l'aparició de les directives `#include` és obligatòria per a totes les funcions de C utilitzades en el programa font.

Acabem de dir que `scanf` és una funció. De moment no s'ha explicat què és, en programació, una funció, però vosaltres ja esteu acostumats a aquest terme en matemàtiques. Us n'oferim una definició:

> Una funció és un conjunt de càlculs que donen un resultat final.

Per tant, `scanf`, a més d'intentar llegir les dades que arriben per teclat, ens dóna un resultat, el qual no és altra cosa que un enter que correspon al nombre de dades llegides per l'entrada estàndard. També s'ha dit més amunt que el llenguatge C no controla si les dades que arriben corresponen al tipus de les variables que les han de rebre; això era tasca del programador. Doncs bé, aquest haurà d'utilitzar el resultat que retorna la funció `scanf` per a efectuar aquest control. Aviat veurem com es pot fer.

Comentem, abans de continuar, en què consisteix el format de lectura que apareix dins la funció `scanf`. El format de lectura interpreta cada dada de l'entrada i està format per caràcters que es poden agrupar en:

1) espais en blanc, que engloba `' '` (espai en blanc), `'\t'` (conjunt de símbols que C interpreta com un tabulador), `'\n'` (conjunt de símbols que C interpreta com un salt de línia, un return o un intro);

2) caràcters ordinaris;

3) especificacions de format (`%d`, `%c`, `%f`, etc.).

El format es llegeix sempre d'esquerra a dreta i cada variable que espera llegir (rebre) una dada ha de tenir la corresponent especificació de format i en el mateix ordre. Posem-ne un exemple:

```
int i, j, k;
char lletra;
float import;
...
scanf ("%d %f %c", &i, &import, &lletra);
scanf ("%d %d", &j, &k);
```

En els exemples anteriors veiem dos casos d'utilització de la funció `scanf`. En el primer, es fa servir per a intentar llegir tres dades omplint les variables `i` (`int`), `import` (`float`) i `lletra` (`char`), mentre que en el segon s'utilitza per a intentar llegir dues dades omplint les variables `j` i `k` (`int`).

Ben segur que us deveu estar preguntant què indiquen les expressions següents:

```
"%d %f %c"
"%d %d"
```

El símbol <return>

El símbol ↵ indica la tecla <entrar>, coneguda normalment per <return> o <intro>.

en el format de lectura. El format de lectura indica al processador com ha d'interpretar les dades que li arribaran. Si un caràcter de l'entrada estàndard no es correspon amb l'entrada especificada pel format, s'avortarà l'entrada de dades i el programa passarà a executar la instrucció següent.

Quan s'executa una instrucció `scanf` com les dels exemples anteriors, l'execució del programa s'atura fins que per l'entrada estàndard (suposem que és el teclat) hi arriben les dades que espera. En cas d'introduir les dades per teclat, aquestes no són assignades immediatament a les variables corresponents; si això succeís, no tindríem possibilitat de corregir una dada equivocada. La realitat és que les dades s'escriuen en una zona de memòria intermèdia anomenada buffer i són enviades al processador quan es prem la tecla <↵>. Aquestes dades, a mesura que s'escriuen, són visualitzades per pantalla amb la finalitat de veure el que l'usuari està introduint.

Quan la unitat central de processament inicia la interpretació de les dades enviades, les ha de poder interpretar tal com especifica el format. Si no ho pot fer, avorta la introducció en la dada on ha fallat i passa a la instrucció següent. Ara bé, les dades que han quedat pendents d'assignar a variables continuen en estat d'espera, de manera que la propera instrucció de lectura no s'aturarà, perquè ja té dades pendents de processar, i com que les dades que s'han de processar no són les que l'usuari hauria volgut introduir, el programa haurà perdut l'oremus i el seu funcionament serà, a partir d'aquest moment, imprevisible. Això no passarà mai si el programador ho controla bé.

La unitat central de processament pot rebre les dades una a una o, d'alguna manera, en seqüència. En tal cas, les rep separades per qualsevol dels caràcters abans agrupats sota la definició genèrica d'espais en blanc, és a dir, l'espai en blanc ' ', el tabulador '`\t`' o el salt de línia '`\n`'. Un espai en blanc abans o després d'una especificació de format fa que `scanf` llegeixi, però no emmagatzemi, tots els caràcters espai en blanc, fins a trobar-ne un diferent de l'espai en blanc.

Posem casos d'introducció de dades basats en els exemples anteriors. Suposem que el buffer d'entrada no conté cap dada pendent de processar i que, per tant, l'ordinador s'atura en el primer `scanf`.

a) Teclegem:

```
5↵
```

i l'ordinador processa aquesta dada, ja que en prémer <↵> s'inicia el procés; l'especificació de format `%d`, que correspon a la primera dada, li indica que s'ha de trobar amb una dada que ha d'interpretar com un `int` i no té cap problema per fer-ho així amb el valor 5 que li ha arribat, el qual assigna a la variable `i`; posteriorment es torna a aturar perquè la mateixa instrucció `scanf` encara té dues variables pendents d'omplir; teclegem:

```
2↵
```

i s'inicia el procés per a aquesta dada; l'especificació de format `%f` que correspon ara, li indica que s'ha de trobar una dada que cal interpretar com a `float` i no té cap problema per fer-ho així amb el valor 2 que li ha arribat, el qual assigna a la variable `import`; posteriorment es torna a aturar perquè la mateixa instrucció `scanf` encara té una variable pendent d'omplir; teclegem:

```
A↵
```

i s'inicia el procés per a aquesta dada; la tercera especificació de format `%c` li indica que s'ha de trobar una dada que cal interpretar com a `char` i no té cap problema per fer-ho així amb el valor A que li ha arribat, el qual assigna a la variable `lletra`. En aquest moment, la unitat central de processament ja ha acabat la primera instrucció `scanf` i n'inicia la segona, davant la qual, com que no ha quedat cap dada pendent de processar, torna a provocar l'aturada del programa en espera que introdueixin més dades, procés que seria semblant al descrit.

b) Teclegem:

```
5 2 A↵
```

i el processador inicia el mateix procés; l'espai que hem deixat entre el 5 i el 2 i entre aquest i la A permeten a la unitat central de processament distingir una dada d'una altra. El resultat final és equivalent al cas anterior. En els dos casos, la funció `scanf` dóna el valor 3 com a resultat perquè ha pogut llegir els tres valors. El programador hauria pogut escriure la instrucció de la manera següent:

```
int r;
...
r = scanf ("%d %f %c", &i, &import, &lletra);
```

i així té a la variable `r` la quantitat de dades realment llegides, i pot actuar de la manera que cregui oportú en cas que no s'hagi pogut completar la instrucció d'entrada.

c) Teclegem:

```
5 hola A↵
```

i l'ordinador, quan repeteix el procés, aconsegueix llegir el 5 com a `int` (`%d`), però falla en la lectura esperada del `float` (`%f`) perquè rep una h, caràcter que no pot identificar amb un `float`. En aquest moment, l'`scanf` acaba donant un 1 per resultat. Queden pendents de procés hola i A, que intentaran ser avaluades en la instrucció següent de lectura; en l'exemple que ens ocupa, la següent instrucció `scanf` espera llegir dos `int` (`%d`), per la qual cosa ja falla en la primera lectura, perquè h no pot ser interpretada com un `int`. S'avorta l'execució d'aquest `scanf`, el qual retorna 0 per resultat (no ha pogut assignar valor a cap de les dues variables). Els valors hola i A continuen pendents de procés.

Evidentment, quan una variable té assignat un valor i mitjançant una instrucció d'entrada se li assigna un de nou, el valor anterior és destruït, ja que passa a ocupar el mateix espai de memòria. En cas, però, que la instrucció de lectura falli, la variable conserva el valor que tenia prèviament.

Per altra banda, dins el format de lectura hi pot haver caràcters ordinaris que no tenen res a veure amb les especificacions de format. Això obliga que l'usuari introdueixi els mateixos caràcters ordinaris, és a dir, la funció `scanf` espera trobar les dades amb els caràcters ordinaris corresponents. Així, la instrucció:

```
int r, e, c;
...
r = scanf("%d euros i %d cèntims", &e, &c);
```

obliga l'usuari a introduir una expressió semblant a la següent:

```
25 euros i 60 cèntims
```

cosa que la funció `scanf` interpreta correctament, ja que esperava trobar una dada que pogués interpretar com un `int` (`%d`), el text "euros i", una dada que pogués interpretar com un `int` (`%d`) i el text "cèntims". Com que això ha estat possible, les variables `r`, `e` i `c` han rebut els valors 2 (nombre de dades llegides), 25 i 60, respectivament.

En l'exemple anterior, si l'usuari hagués teclejat:

```
25 euros amb 60 cèntims
```

la funció scanf només hauria pogut llegir el 25, de manera que la variable r hauria quedat plena amb el valor 1, la variable e amb el valor 25 i la variable c no hauria variat el valor que tenia prèviament. El buffer del teclat hauria quedat ple amb les dades que no han estat processades, és a dir, contindria amb 60 cèntims, els quals intentarien ser processats en la instrucció d'entrada següent.

Vegem, però, com pot ser una especificació de format. La sintaxi d'una especificació de format és la següent:

```
%[*][<ample>][h|l]<tipus>
```

Una especificació de format sempre comença amb `%`. La resta dels elements s'expliquen a les taules 8 i 9.

Taula 8. Utilització dels elements d'especificació de format en la funció `scanf`.

Element	Explicació

*	Un asterisc a continuació del símbol % suprimeix l'assignació de la següent dada d'entrada. Per exemple:
	`scanf ("%d %*c %d %*c, &euros, &cèntims);`
	Una entrada com
	`17 e 45 c`
	produeix l'assignació euros=17 i cèntims=45. Els caràcters e i c són llegits però no assignats.

ample	Màxim nombre de caràcters que s'ha de llegir de l'entrada. Els caràcters en excés no es tenen en compte i queden pendents de processar.
`h` (hac)	Es fa servir com a prefix amb els tipus `d`, `i`, `o`, `x` i `X` per a especificar que el valor és `short int`, o amb `u` per especificar que és un `short unsigned int`
`l` (ela)	Es fa servir com a prefix amb els tipus `d`, `i`, `o`, `x` i `X` per a especificar que el valor és `long int`, o amb `u` per a especificar que és un `long unsigned int`. També s'utilitza amb els tipus `e`, `E`, `f`, `g` i `G` per a especificar que el valor és `double`.
tipus	El tipus determina com ha de ser interpretada la dada d'entrada: com un `char`, com un `int`, com un `float`, etc. Aquest apartat també és obligatori. La taula 9 mostra els tipus que en aquests moments podem utilitzar.

Taula 9. Possibles tipus de dades en l'especificació de format de la funció `scanf`.

Tipus	Tipus de la variable	Entrada esperada
`d`	`int`	Enters amb signe, en base 10.
`o`	`int`	Enters amb signe, en base 8.
`x, X`	`int`	Enters amb signe, en base 16.
`i`	`int`	Enters amb signe en base 10, 16 o 8. Si l'enter comença amb 0 (zero) es pren el valor en octal i si comença amb 0x o 0x el valor es pren en hexadecimal.
`u`	`unsigned int`	Enters sense signe, en base 10.
`f, e, E, g, G`	`float`	Valor real en notació científica: [-]d.dddd[{e|e}[+|-]ddd]
`c`	`char`	Un únic caràcter.

Vegem, a la taula 10, alguns exemples d'utilització de l'especificació de format en la funció `scanf` a partir de les següents declaracions de variables:

```
int ptes; unsigned int edat;
float euros; char sexe;
```

Taula 10. Exemples d'utilització de l'especificació de format en la funció `scanf`.

Instrucció	Entrada	Valor enregistrat a la variable
`scanf ("%d", &ptes)`	25	25
`scanf ("%o", &ptes)`	25	21 (valor decimal de 25 octals)
`scanf ("%i", &ptes)`	25 025 0x25 0X25	25 21 37 37
`scanf ("%x", &ptes)`	25	37 (valor decimal de 25 hex.)
`scanf ("%X", &ptes)`	25	37 (valor decimal de 25 hex.)
`scanf ("%u", &edat)`	15	15
`scanf ("%f", &euros)`	25.60 25.60E1 25.60E–1	25.6 256 2.56
`scanf ("%g", &euros)`	25.60 25.60E1 25.60E–1	25.6 256 2.56
`scanf ("%G", &euros)`	25.60 25.60E1 25.60E–1	25.6 256 2.56
`scanf ("%e", &euros)`	25.60 25.60E1 25.60E–1	25.6 256 2.56
`scanf ("%E", &euros)`	25.60	25.6
`scanf ("%c", &sexe)`	D H	D H

A l'igual que en pseudocodi amb la instrucció `llegir`, la instrucció scanf no es pot utilitzar per a efectuar l'entrada de qualsevol tipus de dada. Així, per exemple, no es pot utilitzar per a introduir valors de tipus enum. Ara bé, com el tipus `enum` guarda valors `int`, sí es possible utilitzar la funció `scanf` per a introduir els corresponents valors `int` en dades de tipus `enum`. (!!)

3.4. Instruccions de sortida

De manera semblant a com introduíem les instruccions d'entrada, els programes gestionen dades que, en algun moment, han de donar a conèixer a l'exterior.

Les instruccions de sortida són les que permeten al programa enviar informació cap a l'exterior.

La informació que el programa envia a l'exterior pot estar confeccionada a partir dels valors emmagatzemats a les variables. Per això, juntament amb el fet que hi ha diferents canals per a enviar informació a l'exterior (la pantalla, els suports magnètics, el mòdem, els altaveus, les impressores, etc.), els llenguatges de programació necessiten instruccions de sortida especialitzades (per tipus de dada i per canal de sortida).

3.4.1. Instruccions de sortida en pseudocodi i en ordinogrames

En pseudocodi, de moment, no ens complicarem la vida i considerarem que la sortida es produeix, únicament, per pantalla.

La instrucció de sortida en pseudocodi és:

```
escriure ( <expressió> [, <expressió> ...] );
```

S'hi observa que en una instrucció escriure el programa pot enviar una o més expressions cap a l'exterior.

A tot el que ja coneixem sobre avaluació d'expressions cal afegir que també és una expressió qualsevol text escrit entre dobles cometes.

A l'apartat "Expressions i ordre d'avaluació d'operadors" s'ha treballat l'avaluació de les expressions.

Per a representar una instrucció de sortida en un ordinograma farem servir un símbol d'operació bàsica de sortida, com es mostra a la figura 8.

Si la sortida es produeix per pantalla podem fer servir el símbol corresponent i tindrem la representació que es mostra a la figura 9.

Figura 8. Representació d'una instrucció de sortida en un ordinograma.

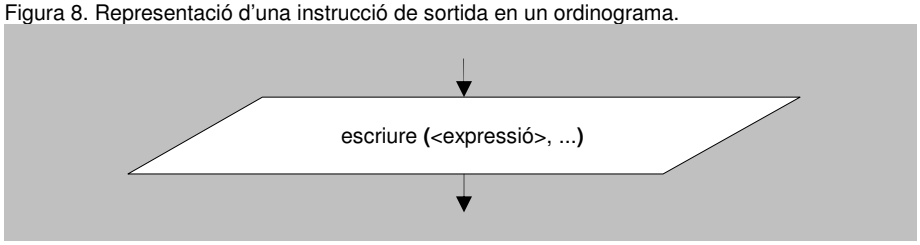
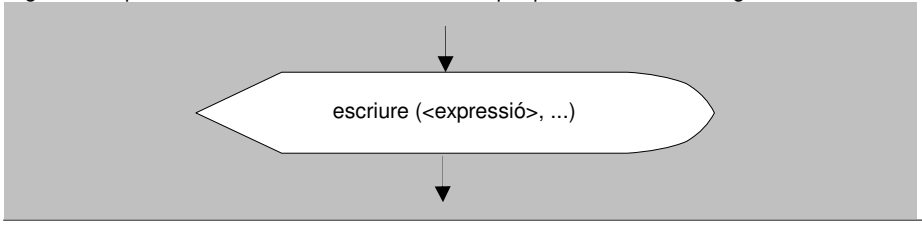


Figura 9. Representació d'una instrucció de sortida per pantalla en un ordinograma.



Per últim, comentar que la instrucció escriure no es pot utilitzar per a efectuar la sortida de qualsevol tipus de dada. Així, per exemple, no es pot utilitzar per a mostrar valors lògics ni valors de tipus enumerats. (!!)

3.4.2. Instruccions de sortida en llenguatge C

En el llenguatge C tenim moltes instruccions corresponents a l'`escriure` del pseudocodi. Ens centrarem, de moment, en una única instrucció: la funció `printf`.

La funció `printf` escriu dades a la sortida estàndard (anomenada `stdout` en llenguatge C i que acostuma a ser la pantalla), d'acord amb el format indicat.

Tal com succeïa amb la funció `scanf`, la funció `printf` admet múltiples variants. Veurem, ara, les que podem fer servir.

La sintaxi és la següent:

```
printf ("<format de sortida>"[, <expressió>, <expressió>, ...] )
```

Observem que és possible que no hi hagi cap expressió. Això és motivat perquè, de manera semblant al pseudocodi, ens podem trobar amb text que ha de ser enviat a l'exterior. En tal cas, el text forma part del format de sortida i, per tant, no serà necessària cap expressió. En aquesta instrucció, entenem per expressió una variable o expressió aritmètica amb un resultat.

I hem comentat que `printf` és una funció, és a dir, ha de retornar un resultat. Així és; dóna un resultat enter igual al nombre de caràcters escrits.

De manera semblant al que es va comentar quan explicàvem la funció `scanf`, la informació que pot necessitar el compilador sobre la funció `printf` es troba a l'arxiu `stdio.h`, de manera que en cas que el compilador es queixi perquè no troba la declaració de la funció `printf`, caldrà incorporar a l'inici del programa font la corresponent `#include`.

En què consisteix el format de sortida que apareix dins la funció `printf`? El format de sortida especifica com serà la sortida. És una seqüència de caràcters formada per:

- caràcters ordinaris (text que apareixerà tal qual);
- especificacions de format (`%d`, `%c`, `%f`, etc.);

- seqüències d'escapament (que ens permetran gestionar salts de línia, avisos d'altaveu, etc.).

El format es llegeix sempre d'esquerra a dreta. Posem-ne un exemple:

```
int a=5;
float b=2.1E2;
...
printf ("El valors de a i b són %d i %f\n",a,b);
```

Per pantalla apareixerà:

```
Els valors de a i b són 5 i 210.000000
```

i el cursor haurà efectuat un salt de línia (provocat per la seqüència d'escapament `\n`).

Vegem la sintaxi d'una especificació de format:

```
%[<flags>][<ample>][.<precisió>][{h|l|L}] <tipus>
```

Una especificació de format sempre comença amb `%`. La resta dels elements s'expliquen a la taula 11.

Taula 11. Utilització dels elements d'especificació de format en la funció `printf`.

Element	Possibilitats	Explicació	
flags	–	Justifica el resultat a l'esquerra dins l'<ample> especificat. Per defecte, la justificació es fa per la dreta.	
	+	Inicia sempre la sortida amb el signe + o – per als valors numèrics. Per defecte només apareix el signe – per als valors negatius.	
	0	Omple la sortida amb zeros no significatius dins l'<ample> especificat.	
	Espai	Inicia sempre la sortida amb un espai en blanc per als valors numèrics positius. S'ignora si s'utilitza juntament amb **+**.	
	#	Funció dependent del <tipus> indicat:	
		Tipus `c`, `d`, `i`, `u`	S'ignora.
		Tipus `o`, `x`, `X`	Inicia la sortida amb 0, 0x, 0X respect.
		Tipus `e`, `E`, `f`, `g`, `G`	Obliga que la sortida contingui un punt decimal.
amplada	Nombre mínim de posicions (caràcters) per la totalitat de la sortida. Si el valor de l'expressió ocupa més caràcters dels especificats, l'amplada és incrementada tant com calgui. Aquest apartat admet la possibilitat d'un * (asterisc), el qual indica que el valor que s'ha de considerar com a <ample> és el resultat de la següent expressió a la llista de les expressions, abans de l'expressió que estem formatant.		
precisió	0	Funció dependent del <tipus> indicat:	
		Tipus d, i, o, u, x, X, c	S'ignora.
		Tipus f, e, E, g, G	Sense punt decimal.

	<n>	A tot estirar, <n> posicions.
	*	El valor que s'ha de considerar com a <precisió> és el resultat de l'expressió següent a la llista de les expressions, abans de l'expressió que estem formatant.
h	S'utilitza com a prefix amb els tipus d, i, o, x i X per a especificar que el valor és short int, o amb u per a especificar que és un short unsigned int.	
l (ela)	S'utilitza com a prefix amb els tipus d, i, o, x i X per a especificar que el valor és long int, o amb u per a especificar que és un long unsigned int. També es fa servir amb els tipus e, E, f, g i G per a especificar que el valor és double.	
L	S'utilitza com a prefix amb els tipus e, E, f, g i G per a especificar que el valor és long double.	
tipus	El tipus determina com ha de ser interpretada la dada de sortida: com un char, com un int, com un float, etc. Aquest apartat també és obligatori. La taula 12 mostra els tipus que en aquests moments podem fer servir.	

Taula 12. Possibles tipus de dades en l'especificació de format de la funció printf.

Tipus	Tipus de la sortida	Sortida efectuada
d, i	int	Enters amb signe, en base 10.
o	int	Enters sense signe, en base 8.
x	int	Enters sense signe, en base 16 (01...abcdef).
X	int	Enters sense signe, en base 16 (01...ABCDEF).
u	int	Enters sense signe, en base 10.
f	double	Valor amb signe en notació[-]dddd.dddd. El nombre de dígits de la part entera depèn de la magnitud del número. El nombre de decimals depèn de la precisió, la qual és, per defecte, 6.
e, E	double	Valor amb signe en notació científica: [-]d.dddd{e|e}[{+|-}]ddd.
g, G	double	Valor amb signe, en format f o e/E segons quin sigui més compacte pel valor i precisió donats.
c	int	Un únic caràcter, corresponent a l'octet menys significatiu del valor considerat com a enter; o sigui, el valor que resulta de col·locar el valor inicial dins l'interval [0...255] "fent tantes voltes" com sigui necessari.

Vegem, a la taula 13, alguns exemples d'utilització de l'especificació de format en la funció printf a partir de les següents declaracions de variables:

```
int ptes=-25; unsigned int edat=20;
float euros=50.5; char sexe='?';
```

Taula 13. Exemples d'utilització de l'especificació de format en la funció printf.

Instrucció	Sortida efectuada
printf("%d",ptes)	-25
printf("%o",ptes)	177747 Com és possible? %o tracta el valor int considerat sense signe; en tal cas, -25 es converteix en 65511 (65536 - 25), ja que l'interval de valors en els enters sense signe és

Instrucció	Sortida efectuada
	[0...65535], i el valor decimal 65511 és, en octal, 177747
`printf("%x",ptes)`	ffe7 Esbrineu el perquè?
`printf( "%u",edat)`	20
`printf("%o",edat)`	24, que és el valor octal de 20 decimal
`printf("%c",sexe)`	?
`printf("%f",euros)`	50.500000
`printf("%.4f",euros)`	50.5000
`printf("%E",euros)`	5.05000E+01
`printf("%.4e",euros)`	5.050e+01
`printf("%g",euros)`	50.5
`printf("%.*E",3,euros)`	5.05E+01

Comentem, a la taula 14, les seqüències d'escapament possibles, anomenades també caràcters de control.

Taula 14. Seqüències d'escapament possibles en l'especificació de format de la funció `printf`

Símbol	Efecte
`\n`	Salta a la línia següent (Line Feed).
`\r`	Retorn de carro (Carriage Return). Provoca anar a l'inici de línia.
`\t`	Tabula la sortida horitzontalment.
`\v`	Tabula la sortida verticalment.
`\b`	Retrocés (espai a l'esquerra).
`\f`	Provoca salt de pàgina (Form Feed).
`\0`	Provoca l'aparició del caràcter nul, és a dir, el caràcter que ocupa el lloc zero a la taula ASCII.
`\a`	Provoca que soni l'altaveu.
`\\`	Provoca la sortida de la barra invertida (\).
`\'`	Provoca la sortida de la cometa simple.
`\"`	Provoca la sortida de les dobles cometes (“).
`%%`	Provoca la sortida del símbol tant per cent (%).

Les seqüències d'escapament cal fer-les servir en els entorns en què es poden fer servir. Dit en altres paraules, un salt de pàgina té sentit quan la sortida s'està produint cap a impressora. Si ho utilitzem en pantalla farà que ens aparegui un símbol estrany, corresponent al símbol que ocupa el lloc 12 (començant a comptar a partir de zero) en el codi ASCII i que correspon al salt de pàgina.

A l'igual que en pseudocodi amb la instrucció `escriure`, la instrucció `printf` no es pot utilitzar per a efectuar la sortida qualsevol tipus de dada. Així, per exemple, no es pot utilitzar per a mostrar valors de tipus enum. Ara bé, com el tipus `enum` guarda valors `int`, sí es possible utilitzar la funció `printf` per a mostrar els corresponents valors `int` emmagatzemats en dades de tipus `enum`. (!!)

3.5. Instruccions d'utilització freqüent

Els programadors solen trobar-se situacions que es repeteixen en els diferents algorismes que han de codificar. Els fabricants dels llenguatges de programació en són conscients i, per facilitar la feina dels programadors i, per tant, agilitar la fase de codificació, incorporen en els llenguatges les instruccions adequades, de manera que el programador no les ha de codificar, sinó que tan sols les ha d'utilitzar.

En realitat no són instruccions, sinó petits programes que farem servir dins els nostres programes i que estan desenvolupats utilitzant les instruccions bàsiques del llenguatge. Però això, a nosaltres, no ens importa. Podem considerar-les instruccions del llenguatge.

Ara bé, no tots els fabricants de llenguatge de programació s'han posat d'acord en desenvolupar instruccions equivalents amb el mateix nom i, per tant, això ens porta a que en utilitzar aquestes instruccions, els nostres programes fonts deixen de ser d'utilització universal. (!!)

Com a exemple d'aquesta situació veurem les instruccions de control de pantalla que el fabricant de llenguatges de programació Borland ha incorporat en la seva saga de productes comercials àmpliament coneguts Turbo C, Turbo C++, Borland C++,... i que en canvi, per exemple, no existeixen en l'entorn de programació de Microsoft Visual C++ ni en el conjunt de compiladors GCC del projecte GNU.
A continuació definirem les instruccions d'ús freqüent que farem servir en pseudocodi i els seus equivalents en llenguatge C.

1) Esborrar la pantalla

Aquesta instrucció, tal com el seu nom indica, deixa la pantalla en blanc (o en negre o verd o taronja, segons el color de la pantalla) i situa el cursor a la primera columna de la primera fila.

- En pseudocodi: `netejar_pantalla`
- En llenguatge C: No hi ha una instrucció específica. Què fem?

– Es pot aconseguir la neteja de pantalla enviant una seqüència de caràcters d'escapament per medi de la funció printf:

```
printf("\033[2J\033[0;0f");
```

En realitat l'anterior sentència `printf` envia dues seqüències de caràcteres d'escapament:

```
printf("\033[2J");   /* Neteja pantalla */
printf("\033[0;0f"); /* Posiciona el cursor a la cantonada superior esquerra */
```

S'entén que la majoria de les vegades la neteja de pantalla comporta el posterior posicionament del cursor a la cantonada superior esquerra.

– Els productes de Borland incorporen una llibreria de funcions per a la gestió de la consola, la definició de les quals es troba a l'arxiu `conio.h` (console input output) que inclourem en el programa font, si és necessari, mitjançant la directriu de precompilació #include.

La utilitat proporcionada per Borland en els seus productes per a la neteja de pantalla és la funció `clrscr()` (provinent de clear screen)

– I en altres entorns? Pot ser que incloguin solucions similars a la funció `clrscr()` de Borland.

2) Esborrar la línia en la que estem situats

Aquesta instrucció, tal com el seu nom indica, deixa la fila on es situat el cursor en blanc i situa el cursor a la primera columna de dita fila.

- En pseudocodi: `netejar_fila`
- En llenguatge C: No hi ha una instrucció específica. Què fem?
  – Es pot aconseguir la neteja de la fila actual enviant una seqüència de caràcters d'escapament per medi de la funció printf:

```
printf ("\r\033[K\r");
```

  – Els productes de Borland incorporen la funció `clreol()` declarada a l'arxiu `conio.h`.

  – I en altres entorns? Pot ser que incloguin solucions similars a la funció `clreol()` de Borland.

3) Posicionar el cursor en pantalla

Aquesta instrucció permet situar el cursor en un lloc determinat de la pantalla, definit per les coordenades de la fila i de la columna.

- En pseudocodi: cursor (<columna>, <fila>)
- En llenguatge C: No hi ha una instrucció específica. Què fem?

  – Es pot aconseguir el posicionament del cursor en pantalla enviant una seqüència de caràcters d'escapament per medi de la funció `printf`:

```
printf("\033[%d;%df",fila,columna);
/* Posiciona el cursor a les coordenades indicades pels valors "fila" i "columna"
considerant que la cantonada superior esquerra correspon a les coordenades (1,1) i
que el valor "columna" indica desplaçament cap a la dreta i el valor "fila" indica
desplaçament cap avall */
```

  – Els productes de Borland incorporen la funció `gotoxy (<columna>, <fila>)` declarada a l'arxiu `conio.h`.

  – I en altres entorns? Pot ser que incloguin solucions similars a la funció `gotoxy()` de Borland.

Posicionament del cursor en pantalla en altres llenguatges

Aquesta instrucció acostuma a estar en tots els llenguatges. En molts, però, primer es posa la <fila> i després la <columna>. També sol variar l'interval de numeració de les files i columnes, ja que a vegades és de 0 a 24 i de 0 a 79.

4) Saltar línia

Aquesta instrucció permet situar el cursor a la primera columna de la fila següent.

- En pseudocodi: `saltar_línia;`
- En llenguatge C: `printf ("\n");`

5) Esperar que l'usuari premi una tecla

Amb aquesta instrucció s'atura l'execució del programa fins que l'usuari premi una tecla.

- En pseudocodi: `esperar_tecla;`
- En llenguatge C: No hi ha una instrucció específica. Què fem?

  – Els productes de Borland incorporen les funcions `getch()` i `getche()` declarades a l'arxiu `conio.h`. Les dues funcions es diferencien perquè la segona mostra per pantalla (en el lloc on és

el cursor) el símbol corresponent a la tecla premuda (la lletra e final indica que es fa echo, és a dir, es mostra per pantalla), mentre que la primera no el mostra.

Totes dues són funcions que retornen un resultat, que no és altre que el caràcter corresponent a la tecla premuda. Aquestes funcions actuen sense buffer, és a dir, la tecla premuda és llegida immediatament sense possibilitat de correcció (a diferència de la funció `scanf`).

– I en altres entorns? Pot ser que incloguin solucions similars a les funcions `getch()` i `getche()` de Borland.

Arxiu `conio.h` per a compiladors gcc

L'arxiu `conio.h` només forma part dels entorns de programació Borland. La utilització d'aquests entorns és, però, tant estesa, que molts programadors pensen que aquest arxiu forma part del llenguatge C estàndard i tenen problemes quan volen canviar de plataforma.

Sempre hi ha la possibilitat, per tal d'aconseguir compatibilitat dels fonts en diferents plataformes, de dissenyar un arxiu `conio.h` que contingui les típiques funcions `clrscr()`, `clreol()`, `gotoxy()`, `getch()`, `getche()` i d'altres definides en l'arxiu `conio.h` original de Borland, de manera que si l'utilitzem, els codis fonts dels programes desenvolupats podran ser vàlids simultàniament en diferents entorns.

Vegeu una versió de l'arxiu `conio.h` per a compiladors `gcc` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit, que facilitarà la possibilitat d'utilitzar les funcions que inclou, en tots els exemples desenvolupats en aquest material.

6) Funcions predefinides

Els llenguatges acostumen a proporcionar als programadors un munt de funcions per a efectuar càlculs estàndards i d'utilitat general en diferents àmbits. No passarem, en aquest moment, a fer una relació exhaustiva de les que ens trobarem, ja que això també depèn del llenguatge. En pseudocodi, a mesura que avancem, n'introduirem i, paral·lelament, les treballarem en el llenguatge C.

A tall informatiu, sapigueu que podreu trobar:

- Funcions numèriques per a calcular valors absoluts, logaritmes, exponencials, potències, arrels, etc.

- Funcions trigonomètriques per a calcular les raons trigonomètriques: sinus, cosinus, tangent, arcsinus, arccosinus, arctangent, sinus hiperbòlics, cosinus hiperbòlics, tangents hiperbòliques, etc. No us preocupeu si no recordeu la trigonometria. Per a desenvolupar programes de gestió no és fàcil que la necessiteu. Ara bé, això no vol dir que no l'hagueu de conèixer.

- Funcions alfanumèriques per a treballar amb expressions numèriques i de text.

3.6. Exemples d'utilització d'instruccions seqüencials bàsiques

Tot seguit us exposem alguns exemples per a il·lustrar el que heu après fins ara.

1) Fer un programa que calculi la superfície d'un rectangle

Primer de tot hem de saber què vol dir "calcular la superfície d'un rectangle". És a dir, en aquest cas, l'analista ha de tenir coneixements de matemàtiques per poder dissenyar l'algorisme corresponent. Fem l'anàlisi funcional. No entrem a detallar el resultat d'aquesta anàlisi perquè ho sabem fer des de fa molt de temps, quan anàvem a escola.

Flexibilitat de l'analista

L'analista ha de ser una persona amb capacitat per a introduir-se en qualsevol camp. Si es tracta de desenvolupar una aplicació informàtica per gestionar la facturació d'una empresa, caldrà conèixer una mica com és el sector en el qual es mou l'empresa. No és el mateix una facturació per a una empresa que es dedica a la fabricació de mitjons que per a una empresa que es dedica a la fabricació de cotxes, tot i que la part legal de les dues facturacions deu ser molt semblant.

Suposant que ja sabem què vol dir "calcular la superfície d'un rectangle", entrem a efectuar l'anàlisi orgànica per a poder descriure l'algorisme que ens permeti resoldre el problema. Posem-nos en el lloc de l'ordinador i el primer que fem és preguntar a l'usuari quant val la base i quant val l'altura. Abans, però, l'hem informat que ens ha de donar els dos valors en les mateixes unitats.

Quan l'usuari ens hagi donat els dos valors els multiplicarem i informarem l'usuari del resultat, que és la superfície del rectangle.

Per poder efectuar la multiplicació dels dos valors, necessitem haver-los guardat en algun lloc. La persona utilitza la seva memòria; l'ordinador també, mitjançant les variables. Per tant, necessitarem dues variables per a emmagatzemar els valors de la base i de l'altura. Sembla evident que hauran de ser variables numèriques. Millor que siguin reals per a poder tractar valors decimals.
Per a poder informar l'usuari del valor de la superfície és necessari que aquest valor estigui en algun lloc. Per tant, necessitem una altra variable per a emmagatzemar el resultat del producte. Així doncs:

```
programa superfície_rectangle és
   var
      base, altura, superfície : real;
   fivar
   netejar_pantalla;
   escriure ("Haurà d'introduir els dos valors amb les mateixes unitats");
   saltar_línia;
   escriure ("Introdueixi la base:");
   llegir (base);
```

```
   saltar_línia;
   escriure ("Introdueixi l'altura");
   llegir (altura);
   saltar_línia;
   superfície = base * altura;
   escriure ("La superfície del rectangle que té per base ",base," unitats i per altura
", altura, " unitats és ", superfície, "unitats quadrades.");
   saltar_línia;
fiprograma
```

- La darrera instrucció escriure és molt llarga i no ens cap en una línia. Podem optar pel que tenim a l'exemple, és a dir, per anar escrivint de manera que, si l'editor ho mostra automàticament en una altra línia, és decisió seva, però nosaltres, en realitat, ho hem escrit tot en una línia. Però també tenim una altra possibilitat, que és tallar la línia en tantes línies com ens sembli convenient, acabant cada línia amb el símbol \, el qual indicarà al compilador que aquella línia i la següent han de ser considerades com una sola línia. Aquest símbol no pot estar (en pseudocodi) dins de cap text entre dobles cometes, ja que en aquest cas no tindria aquesta interpretació i, segurament, ens portaria problemes.

Hem utilitzat el símbol \ perquè és el símbol que utilitza el llenguatge C. En altres llenguatges no es pot fer servir, o utilitzen altres marques.

- La darrera instrucció escriure conté expressions de diferents tipus separades per comes. Si l'usuari ha introduït 10 com a resposta a la pregunta "Introdueixi la base:" i 20 com a resposta a la pregunta "Introdueixi l'altura", el programa haurà emmagatzemat 200 a superfície i, per tant, la sortida per pantalla del darrer escriure és:

```
La superfície del rectangle que té per base 10 unitats i per altura 20 unitats és 200
unitats quadrades.
```

- Ens podem estalviar la variable superfície. Aquesta variable només s'utilitza per a rebre el resultat del producte de la base per l'altura i, posteriorment, ser enviat a la sortida estàndard. La instrucció escriure permet, en la majoria de llenguatges, la inclusió d'expressions numèriques, de manera que s'avaluen abans de produir-se la sortida i, el que s'escriu, és el resultat.

Nova versió amb les observacions efectuades:

```
programa superfície_rectangle és
   var
      base, altura : real;
   fivar
   netejar_pantalla;
   escriure ("Haurà d'introduir els dos valors amb les mateixes unitats");
   saltar_línia;
   escriure ("Introdueixi la base:");
   llegir (base);
   saltar_línia;
```

```
   escriure ("Introdueixi l'altura");
   llegir (altura);
   saltar_línia;
   escriure  ("La superfície del rectangle que té per base ", base, " unitats i ",\
      "per altura ", altura, " unitats és ", base * altura, "unitats quadrades.");
   saltar_línia;
fiprograma
```

Versió en llenguatge C:

```
/* u1n3p01.C */

#include <stdio.h>
#include <conio.h>

void main()
{
   float base, altura;
   clrscr();
   printf ("Haurà d'introduir els dos valors amb les mateixes unitats\n\n");
   printf ("Introdueixi la base: ");
   scanf ("%f",&base);
   printf ("Introdueixi l'altura: ");
   scanf ("%f",&altura);
   printf ("La superfície del rectangle que té per base %g unitats i per \
altura %g unitats és %g unitats quadrades\n", base, altura, base*altura);
}
```

Trobareu l'arxiu `u1n3p01.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Hem de tenir en compte el següent:

- Després de cada scanf no hem posat un salt de línia i, si ho executem, veiem que el fa. Com és possible? Això és així perquè l'usuari, per validar l'scanf, ha de prémer un <return>, el qual provoca el salt de línia.

- En aquesta versió en programa C no hi ha cap control sobre els valors introduïts per l'usuari. És a dir, si l'usuari entra caràcters quan l'ordinador espera reals, pot passar qualsevol cosa. En aquests moments no estem encara en disposició de poder-ho controlar, tot i que caldrà utilitzar el valor que retorna la funció scanf relatiu al nombre de dades que hagi pogut llegir.
- En el darrer printf hem obligat a efectuar la sortida dels valors reals amb l'especificació de format %g. D'aquesta manera, el programa decideix, en temps d'execució, quina és la manera més compacta d'escriure el valor, cosa que dependrà de cada execució.

- Hem utilitzat les directrius de precompilació #include per a indicar al compilador la sintaxi de les funcions de C utilitzades. Això no és necessari en tots els entorns, però és millor acostumar-s'hi.

- Aquest exemple ens serveix per a introduir, en C, el significat del símbol \ a final de línia, semblant al que hem indicat en pseudocodi.

  Quan s'escriu un programa font en llenguatge C, utilitzant un editor de textos, les línies acostumen a tenir una llargada important (de prop de 250 caràcters), la qual segurament no farem servir en la seva totalitat. Però, i si necessitéssim una línia més llarga? El símbol \ a final de línia permet indicar al programa traductor que el text que hi ha a la línia següent és continuació de la línia actual.
  En el llenguatge C, el símbol \ es pot utilitzar enmig de textos entre dobles cometes (és a dir, un fragment del text en una línia i la continuació en l'altra). No es pot, però, trencar noms de variables, paraules reservades, funcions, etc.

  En l'exemple anterior, podeu observar que hem fet servir el símbol \ a final d'una línia que contenia la instrucció printf. Això vol dir que la línia següent forma part de la mateixa instrucció. Quan vosaltres escriviu el programa, veureu que no us és necessari aquest símbol, però per poder incloure el codi font en aquest material, ha calgut trencar la instrucció.

  El símbol \ a final de línia

  La utilització del símbol \ a final de línia, tot i no ser necessària quasi mai, és convenient per a la impressió dels codis font. Quan s'estan codificant programes amb força línies de codi i alguna cosa no acaba de funcionar, és bastant difícil seguir el programa en la pantalla de l'ordinador, ja que té un nombre bastant petit de línies. En tal cas, el programador acostuma a obtenir una còpia impresa del codi font i és més fàcil fer el seguiment.

  Les còpies impreses, però, tenen una limitació en l'amplada de línia. Per tant, és bo acostumar-se a no treballar amb línies gaire llargues (tot i que l'editor ens ho permeti) pensant en la impressió del codi font. En aquest cas és necessari utilitzar el símbol \ per trencar línies.

2) Fer un programa que calculi la longitud d'una circumferència i la superfície i el volum de cercle i esfera corresponents

En aquest cas, l'anàlisi funcional també ha estat fàcil, ja que recordem (i si no les recordàvem no ens costa res cercar-les en algun llibre) les fórmules per als càlculs esmentats:

Longitud circumferència = $2 * \pi * R$.
Superfície cercle = $\pi * R^2$.
Volum esfera = $4 * \pi * R^3 / 3$.

Doncs bé, en l'etapa de l'anàlisi orgànica observem que serà necessari disposar d'una variable de tipus real on es pugui recollir el radi del cercle per al qual l'usuari vol calcular la superfície.

A més, aquí apareix el valor π que, com molt bé sabeu, representa una constant amb un nombre infinit de xifres decimals. Doncs bé, decidim utilitzar una constant per a emmagatzemar aquest valor i fer servir, quan sigui necessari, la constant. En aquest programa, la constant esmentada caldrà utilitzar-la en tres càlculs. Havent-la declarat com a constant evitem haver d'escriure tres vegades el valor concret 3.1415..., amb la qual cosa aconseguim dues fites: evitar errors d'escriptura del valor en els diferents indrets (tres) en què s'utilitza i facilitar la possibilitat d'efectuar els càlculs amb més precisió (afegint-hi més xifres decimals), ja que només caldrà modificar el valor de la constant en la seva declaració.

Per tant, en pseudocodi:

```
programa LSV_radi és
   const
      PI = 3.1415;
   ficonst
   var
      radi : real;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi el radi:");
   llegir (radi);
   saltar_línia;
   escriure ("Donat el radi ", radi, " tenim:");
   saltar_línia;
   escriure ("Longitud circumferència: ", 2 * PI * radi);
   saltar_línia;
   escriure ("Superfície cercle: ", PI * radi * radi);
   saltar_línia;
   escriure ("Volum esfera: ", 4*PI * radi * radi * radi / 3);
   saltar_línia;
fiprograma
```

La versió en llenguatge C:

```
/* u1n3p02.c */

#include <stdio.h>
#include <conio.h>

main()
{
   const float PI=3.1415;
   float radi;
   clrscr();
   printf ("Introdueixi el radi: ");
   scanf ("%f",&radi);
   printf ("Donat el radi %g tenim:\n\n", radi);
   printf ("Longitud de la circumferència: %10.3f\n", 2*PI*radi);
   printf ("Superfície del cercle.....: %10.3f\n", PI*radi*radi);
   printf ("Volum de l'esfera.....: %10.3f\n", 4*PI*radi*radi*radi/3);
}
```

Fixeu-vos en els punts següents:

- Amb l'especificació de format %10.3f per als tres resultats, hem obligat que en tots la sortida tingui un mínim de 10 caràcters i sempre amb tres decimals. Quan no arriba a 10 caràcters, es justifica amb espais blancs per l'esquerra. En cas que els resultats fossin superiors a 10 caràcters, el programa agafaria l'espai necessari.

- Recordeu que en C es poden tenir constants simbòliques que utilitza el precompilador per a substituir totes les aparicions en el programa font i que, per tant, en temps d'execució no ocupen memòria. En la versió anterior, la constant definida no és constant simbòlica.

Trobareu l'arxiu `u1n3p02.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Nova versió en llenguatge C utilitzant constant simbòlica:

```
/* u1n3p03.c */

#define PI 3.1415

#include <stdio.h>
#include <conio.h>

main()
{
   float radi;
   clrscr();
   printf ("Introdueixi el radi: ");
   scanf ("%f",&radi);
   printf ("Donat el radi %g tenim:\n\n", radi);
   printf ("Longitud de la circumferència: %10.3f\n", 2*PI*radi);
   printf ("Superfície del cercle........: %10.3f\n", PI*radi*radi);
   printf ("Volum de l'esfera............: %10.3f\n", 4*PI*radi*radi*radi/3);
}
```

Trobareu l'arxiu `u1n3p03.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.