Estructures de control

2

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

A la unitat didàctica "Introducció a la programació" classificàvem els algorismes, segons el flux d'execució, en seqüencials, condicionals i repetitius, tot i que dèiem que en realitat molt pocs algorismes es podien classificar en un dels tres tipus anteriors sinó que acostumen a ser una seqüència lineal d'accions algunes de les quals són repetitives i d'altres, condicionals. També hem introduït els tipus de dades simples (sense els quals poca cosa podem fer) i les instruccions seqüencials bàsiques; també hem dissenyat algorismes seqüencials per a resoldre algun problema, com ara el del càlcul de la superfície d'un rectangle i altres.

Però la veritat és que amb el flux seqüencial pocs problemes podem resoldre, ja que la majoria de programes han de repetir una sèrie d'instruccions fins que es compleixi o es deixi de complir una sèrie de condicions; en molts punts han de prendre decisions respecte a si cal seguir l'execució per un camí o un altre. Necessitem, per tant, mecanismes que ens permetin programar en aquestes situacions. Apareixen així les estructures de control condicionals i repetitives, objecte d'estudi d'aquesta unitat didàctica. S'anomenen estructures de control perquè permeten controlar el flux del programa en temps d'execució.

En el nucli d'activitat "Estructures concidionals " veurem les sentències que proporciona la programació estructurada per a prendre decisions respecte a si cal seguir l'execució per un camí o un altre.

En el nucli d'activitat "Estructures repetitives" veurem les sentències que proporciona la programació estructurada per a repetir una sèrie d'instruccions fins que es compleixi o es deixi de complir una sèrie de condicions.

En ambdós nuclis d'activitat presentarem les sentències de control que els diferents llenguatges que suporten programació estructurada i modular faciliten i dissenyarem programes utilitzant el pseudocodi. Veurem, també, quines són les sentències que proporciona el llenguatge C i, per cada programa dissenyat en pseudocodi en realitzarem la corresponent codificació en llenguatge C per tal d'assolir el codi executable i poder procedir a la seva execució i comprovació de la correctesa de funcionament.

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. Dissenyar algorismes en pseudocodi per a resoldre problemes amb la utilització de les sentències de control condicionals i iteratives.

2. Identificar i aplicar la representació gràfica per a les sentències de control condicionals i iteratives.

3. Aplicar les sentències de control condicionals i iteratives en la codificació de programes en llenguatge C.

4. Fer servir de manera adequada el depurador del llenguatge C per al seguiment de l'execució dels programes que continguin sentències de control condicionals i iteratives.

1. Estructures condicionals

En aquest apartat estudiareu les estructures condicionals simples, compostes i múltiples.

Les estructures condicionals o alternatives permeten prendre decisions sobre quin conjunt d'instruccions cal executar en un punt del programa.

Tota estructura condicional es basa en l'avaluació d'una expressió que ha de donar un resultat lògic: cert o fals. Anomenarem aquesta expressió condició lògica de l'estructura. (!!)

La condició lògica es basa, normalment, en els valors emmagatzemats a les variables, de manera que el resultat dependrà de l'execució concreta. El disseny no seria correcte si una estructura alternativa donés sempre el mateix resultat: quin sentit tindria?

Exemple de condició lògica

Imaginem-nos una persona que fa el comentari següent:

"Si plou o no plou, agafa el paraigües. En cas contrari, no l'agafis".

Oi que és clar que la persona que rep el missatge ha d'agafar sempre el paraigües? Llavors, oi que és una pèrdua de temps l'enunciat anterior? Seria millor dir:

"Agafa el paraigües".

1.1. Estructura condicional simple

Un dels mecanismes que ens permeten programar en situacions en què cal prendre decisions respecte a la instrucció següent és l'estructura condicional simple.

L'estructura condicional simple permet controlar el fet que s'executi un conjunt d'instruccions, si i només si es compleix la condició lògica (o sigui, condició lògica avaluada amb el valor cert).

A continuació veurem com s'expressa l'estructura condicional simple en pseudocodi, ordinogrames i llenguatge C.

1.1.1. Estructura condicional simple en pseudocodi i ordinogrames

La sintaxi d'aquesta sentència de control en pseudocodi és la següent:

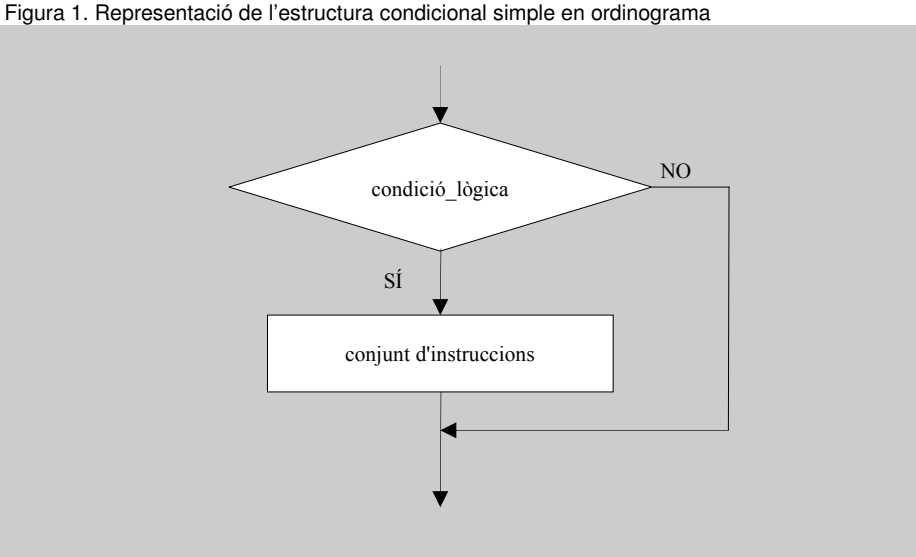
```
si <condició_lògica> llavors
   <conjunt d'instruccions a executar>
fisi
```

Cal interpretar el funcionament d'aquesta sentència com segueix. Quan en temps d'execució el programa arriba a la paraula reservada `si`, avalua la condició lògica, de manera que si el resultat és `cert`, s'executa el conjunt d'instruccions que hi ha entre la paraula reservada `llavors` i la paraula reservada `fisi`. En cas que la condició lògica sigui avaluada com a `fals` o, després d'haver executat el conjunt d'instruccions, en cas d'haver estat avaluada com a `cert`, el programa continua l'execució amb la instrucció que hi ha immediatament després a la paraula reservada `fisi`.

A l'hora d'escriure les instruccions és convenient (no obligatori) sagnar cap a la dreta el contingut de les instruccions que cal executar en cas que la condició lògica sigui avaluada com a certa. D'aquesta manera es facilita el seguiment del programa, ja que visualment s'identifiquen ràpidament totes les instruccions que, d'alguna manera, formen un bloc.

Per a representar les condicions en ordinogrames s'utilitza un símbol en forma de rombe del qual acostumen a sortir dues fletxes (per dues puntes) corresponents al camí que cal seguir en cas que la condició sigui avaluada com a certa o com a falsa. La figura 1 mostra la representació de l'estructura condicional simple en ordinograma.

Figura 1. Representació de l'estructura condicional simple en ordinograma



1.1.2. Estructura condicional simple en llenguatge C

En la majoria de llenguatges, l'estructura condicional simple és molt semblant a la que hem vist en pseudocodi, i el llenguatge C, en això, no n'és una excepció. Així doncs, la sintaxi és:

```
if (<condició_lògica>)
{  <conjunt d'instruccions a executar>; }
```

Cal tenir en compte els punts següents:

- És obligatori emmarcar la condició lògica entre parèntesis.
- No hi ha paraules reservades equivalents a les paraules `llavors` i `fisi` del pseudocodi. Els símbols {} emmarquen les instruccions incloses entre el `llavors` i el `fisi`.
- Si el conjunt d'instruccions que s'ha d'executar consisteix en una única instrucció, els símbols {} es poden estalviar. Per tant, si després de la condició lògica no hi ha text emmarcat entre {}, vol dir que el cos d'instruccions que s'ha d'executar està format per una única instrucció.
- En llenguatge C no existeix el valor lògic i, per tant, la condició lògica pot ser una expressió de qualsevol tipus que serà avaluada com a certa si no és zero.
- Un típic error de programador novell (i també dels no novells, però que han programat amb altres llenguatges), és escriure una expressió semblant a:

```
int a=5, b=3;
...
/* suposem que les variables a i b no han estat modificades */
printf ("a = %d --- b = %d\n", a, b);
if (a = b) printf ("a i b són iguals\n");
/* observeu que no hi ha claus; per tant, en cas que l'expressió sigui avaluada com a certa,
s'executarà l'única instrucció printf ("a i b...*/
printf ("a = %d --- b = %d\n", a, b);
...
```

Què us sembla? Què escriu l'ordinador? Deveu estar d'acord que escriu:

```
a = 5 --- b = 3
a = 5 --- b = 3
```

O sigui, que la condició de l'`if` ha estat avaluada com a falsa perquè, evidentment, 5 i 3 no són iguals, i per tant no s'ha escrit el text a i b són iguals.

Si l'anterior ha estat el vostre raonament, sou novells en C. Molt malament. Però tranquils, això també ens va passar a nosaltres...

El problema està en la manera com s'ha escrit la condició: no s'ha utilitzat l'operador d'igualtat (doble signe igual) sinó l'operador d'assignació (un signe igual). De fet, alguns compiladors adverteixen un possible error i acostumen a donar un avís de perill (`warning`) si no tenim desactivats aquest tipus d'avís. Pot donar un error similar a:

```
Possibly incorrect assignment in function main
```

és a dir, assignació probablement incorrecta en el programa, i ens marca la condició a = b.

Tipologia d'avisos

En el procés de compilació hi ha dos tipus d'avisos:

- Avisos de perill (anomenats `warning`), que ens alerten de situacions no gaire clares des del punt de vista del compilador, però que poden ser acceptades i, ja sabeu que davant el dubte, C opta per obeir a cegues el programador.
- Avisos d'error que no fan possible aconseguir el programa objecte.

Un bon consell: no desactiveu els avisos de perill i, si podeu, modifiqueu el codi per a eliminar-los. El compilador C és molt intel·ligent, no ho poseu en dubte!

Fixem-nos bé com s'executa el programa. En el primer `printf` no hi ha dubte. S'imprimeix els valors de les variables `a` i `b`, que són 5 i 3:

```
a = 5 --- b = 3
```

Però quan arriba a `a = b` es produeix l'assignació del valor de `b (3)` a la variable `a`, de manera que `a` i `b` tenen el mateix valor `3`. A més, aquest valor (no zero) és el que utilitza C com a resultat de l'expressió que l'`if` ha d'avaluar. I ja sabem que una expressió amb valor diferent de zero és avaluada com a certa. En conseqüència, s'executa el `printf` de l'`if` i apareix:

```
a i b són iguals
```

Posteriorment, la següent instrucció que s'executa és l'altre `printf`, el qual s'executarà sigui quina sigui l'avaluació de la condició i, en aquest cas, escriu:

```
a = 3 --- b = 3
```

Penseu el que passaria si `b` hagués estat inicialitzada amb el valor `zero`.

1.1.3. Exemple d'utilització d'estructura condicional simple

Volem efectuar un programa que demani a l'usuari la seva edat i que, en cas de ser major d'edat, l'informi que ja pot votar. En cas que no ho sigui, no ha de fer res d'especial. Posteriorment, el programa ha d'informar l'usuari de l'edat que ha rebut.

Anàlisi funcional

Cal esbrinar quan s'aconsegueix la majoria d'edat. Arribem a la conclusió que això depèn de cada estat. Suposarem que en el nostre estat en aquests moments això succeeix als divuit anys.

Anàlisi orgànica

Cal demanar l'edat a l'usuari i memoritzar-la. Per a fer això necessitem una variable natural. Una vegada tinguem l'edat, comprovarem si és major o igual a divuit i, en tal cas, l'informarem de la possibilitat de votar.

Posteriorment, en qualsevol cas, li farem saber, per pantalla, l'edat que ens ha introduït.

Prenem la decisió de considerar el valor 18 (edat límit per a la majoria d'edat) com una constant i així el nostre programa és fàcilment modificable en cas de modificació legal respecte de la majoria d'edat i fàcilment transportable a estats amb diferent tipus d'edat, ja que les modificacions que s'haurien d'efectuar serien mínimes (modificar la inicialització de la constant).

Escrivim l'algorisme en pseudocodi:

```
programa edat_1 és
   const majoria = 18;
   ficonst
   var   edat = natural;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi la seva edat, si li plau:");
   llegir (edat);
   saltar_línia;
   si edat >= majoria llavors
      escriure ("Caram, és major d'edat! Ja pot votar!");
      saltar_línia;
   fisi
```

```
   escriure ("L'edat introduïda és ",edat," anys");
   saltar_línia;
fiprograma
```

Codifiquem-lo en llenguatge C:

```
/* u2n1p01.C */

#include <stdio.h>
#include <conio.h>
#define MAJORIA 18

void main(void)
{
   unsigned int edat;
   clrscr();
   printf ("Introdueixi la seva edat, si li plau:");
   scanf ("%u",&edat);
   if (edat>=MAJORIA)
      printf ("Caram, és major d'edat! Ja pot votar!\n");
   printf ("L'edat introduïda és %u anys",edat);
}
```

Trobareu l'arxiu `u2n1p01.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

El contingut de l'apartat `llavors` en la instrucció `si...fisi` estava format per dues instruccions en pseudocodi, les quals han esdevingut una de sola en codificar en C. D'altra banda, hauríem pogut mantenir també dues instruccions i llavors hauria estat imprescindible emmarcar-les entre {}. O sigui:

```
   if (edat>=MAJORIA)
   {  printf ("Caram, és major d'edat! Ja pot votar!");
      printf ("\n");
   }
```

Tal com s'ha dit en pseudocodi, és convenient desplaçar cap a la dreta el bloc d'instruccions que forma el `llavors`. En llenguatge C aquest bloc va obligatòriament entre {} si n'hi ha més d'una, la qual cosa també ajuda a identificar visualment el bloc d'instruccions.

1.2. Estructura condicional doble

L'estructura condicional doble afegeix un grau de complexitat a la que hem vist fins ara.

L'estructura condicional doble permet controlar el fet que s'executi un conjunt d'instruccions, si i només si es compleix la condició lògica, i que se n'executi un altre conjunt, si i només si no es compleix la condició lògica.

Igual que en l'estructura condicional simple, veurem com expressar l'estructura condicional doble en pseudocodi, ordinograma i llenguatge C.

1.2.1. Estructura condicional doble en pseudocodi i ordinogrames.

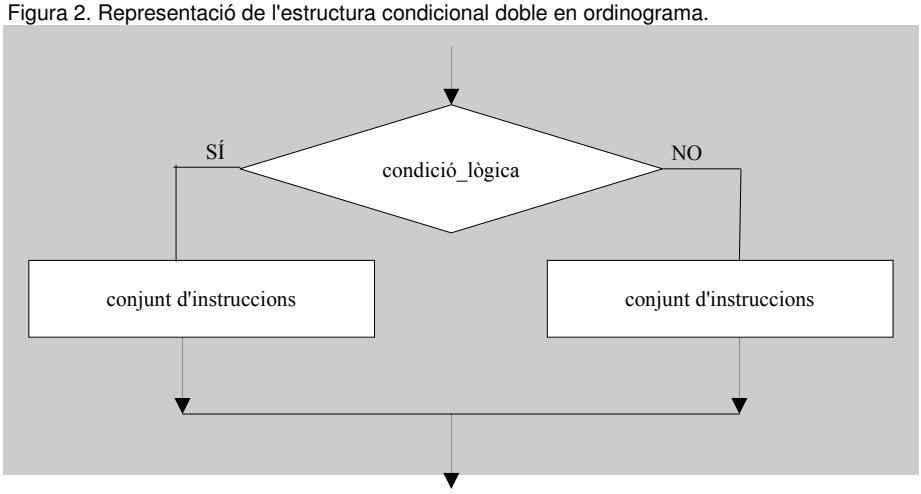
La sintaxi d'aquesta sentència de control en pseudocodi és:

```
si <condició_lògica> llavors
   <conjunt d'instruccions a executar>
sinó
   <conjunt d'instruccions a executar>
fisi
```

Cal interpretar el funcionament d'aquesta sentència com segueix. Quan en temps d'execució el programa arriba a la paraula reservada `si`, avalua la condició lògica, de manera que si el resultat és `cert`, s'executa el conjunt d'instruccions que hi ha entre la paraula reservada `llavors` i la paraula reservada `sinó` i en cas que el resultat sigui `fals` s'executa el conjunt d'instruccions que hi ha entre la paraula reservada `sinó` i la paraula reservada `fisi`. En qualsevol cas, posteriorment a l'execució del bloc d'instruccions que correspongui (només un!) el programa segueix l'execució de les instruccions que apareixen després de la paraula `fisi`.

La figura 2 mostra la representació de l'estructura condicional doble en ordinograma.

Figura 2. Representació de l'estructura condicional doble en ordinograma.



1.2.2. Estructura condicional doble en llenguatge C

El llenguatge C també suporta l'estructura condicional doble, amb la sintaxi següent:

```
if (<condició_lògica>)
{  <conjunt d'instruccions a executar si és certa>; }
else
{  <conjunt d'instruccions a executar si és falsa>: }
```

Cal tenir en compte els punts següents:

- És obligatori emmarcar la condició lògica entre parèntesis.
- No hi ha paraules reservades equivalents a les paraules `llavors`, i `fisi` del pseudocodi. En canvi, la paraula `sinó` té l'equivalent `else`. Els símbols {} emmarquen les instruccions previstes en els espais `llavors ... sinó` i `sinó ... fisi`.
- Si un conjunt d'instruccions que s'ha d'executar consisteix en una única instrucció, els símbols {} corresponents es poden estalviar.

1.2.3. Exemple d'utilització d'estructura condicional doble

Volem un programa que demani a l'usuari la seva edat, i que, en cas de ser-ne major, l'informi que ja pot votar i en cas que no ho sigui, l'informi que encara no pot votar. El programa ha d'incloure en la informació que dóna a l'usuari l'edat que aquest hi ha introduït.

Anàlisi funcional

Cal esbrinar quan s'aconsegueix la majoria d'edat. Arribem a la conclusió que això depèn de cada estat. Suposarem que en el nostre estat en aquests moments això succeeix als divuit anys.

Anàlisi orgànica

Cal demanar l'edat a l'usuari i memoritzar-la. Per a fer això necessitem una variable natural. Una vegada tinguem la seva edat, comprovarem si és major o igual a 18 i l'informarem de la manera adequada.

Escrivim l'algorisme en pseudocodi:

```
programa edat_2 és
   const majoria = 18;
   ficonst
```

```
   var edat = natural;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi la seva edat, si li plau:");
   llegir (edat);
   saltar_línia;
   si edat >= majoria llavors
      escriure ("Té ", edat, " anys. És major d'edat! Ja pot votar!");
   sinó
      escriure ("Té ", edat, " anys. És menor d'edat! No pot votar!");
   fisi
   saltar_línia;
fiprograma
```

Quan s'executa el programa, es preguntarà una vegada si l'edat introduïda per l'usuari, emmagatzemada a la variable `edat`, és major o igual al valor emmagatzemat a la constant `majoria`, i en funció de la resposta, executarà un `escriure` o un altre. Però només fa una pregunta! Seria un mal disseny substituir el fragment d'algorisme corresponent al `si...fisi` per:

```
   si edat >= majoria llavors
      escriure ("Té ", edat, " anys. És major d'edat! Ja pot votar!");
   fisi
   si edat < majoria llavors
      escriure ("Té ", edat, " anys. És menor d'edat! No pot votar!");
   fisi
```

En aquest cas, el programa fa dues preguntes, cada una de les quals és contrària a l'altra. Sigui quina sigui la resposta introduïda per l'usuari, el programa farà les dues preguntes, però una és innecessària. Evidentment, el programa funciona, ja que dóna el resultat esperat. Però està mal dissenyat.

L'objectiu d'aquest crèdit és aprendre a programar, i això implica dissenyar algorismes eficients. Quan és eficient un algorisme? (!!)

> Un algorisme és eficient si en el seu disseny no hi ha parts inútils.

Algorismes eficients

Quan acabeu aquest crèdit, haureu de ser capaços d'aplicar els algorismes més eficients, d'entre els estudiats, per a resoldre els problemes que se us plantegin.

Donat un problema hi pot haver diferents algorismes que aportin la corresponent solució. Possiblement un serà més eficient que l'altre. Caldrà aplicar el més eficient. Aquesta feina correspon a l'analista orgànic.

Per tant, cal distingir bé quin és el millor algorisme per a la resolució d'un problema i aconseguir que el disseny de l'algorisme sigui eficient. La darrera situació presentada és un clar exemple de disseny no eficient.

Codifiquem el disseny correcte en llenguatge C:

```
/* u2n1p02.C */

#include <stdio.h>
#include <conio.h>
#define MAJORIA 18

void main(void)
{
   unsigned int edat;
   clrscr();
   printf ("Introdueixi la seva edat, si li plau:");
   scanf ("%u",&edat);
   if (edat>=MAJORIA)
      printf ("Té %u anys. És major d'edat! Ja pot votar!\n", edat);
   else
      printf ("Té %u anys. És menor d'edat! No pot votar!\n", edat);
}
```

Trobareu l'arxiu `u2n1p02.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

1.3. Imbricació d'estructures condicionals simples i dobles

En les estructures condicionals, tant si són simples com si són dobles, es poden donar imbricacions.

> Es produeix imbricació d'estructures condicionals simples i dobles quan dins d'una estructura condicional si...llavors...[sinó...]fisi hi ha una altra estructura condicional simple o doble.

És a dir, situacions com:

```
si condició_1 llavors
   instrucció;
   ...
   si condició_2  llavors
      instrucció;
      ...
   sinó
      instrucció;
      ...
   fisi
   ...
sinó
   instrucció;
   ...
   si condició_3 llavors
      ...
   fisi
fisi
```

Tornant al comentari fet amb anterioritat referent a que és important no fer preguntes inútils i ho hem aplicat a la conveniència d'utilitzar l'estructura condicional doble abans que fer servir dues estructures condicionals simples de manera seqüencial, la generalització del comentari consisteix a notar que en moltes ocasions caldrà imbricar estructures condicionals dins d'altres estructures condicionals, per donar claredat als algorismes i permetre la resolució del problema amb les mínimes instruccions necessàries.

Cal, però, anar amb compte i controlar, per cada `si`, els corresponents `llavors`, `sinó` i `fisi`. Aquesta tasca se simplifica si se segueix el consell de sagnar els conjunts d'instruccions cap la dreta, tal com es pot observar en l'exemple anterior, mantenint les paraules `si`, `sinó` i `fisi` d'una mateixa estructura condicional a una mateixa distància del marge esquerre.

Els llenguatges de programació acostumen a permetre la imbricació de les estructures condicionals. Hi pot haver, però, l'excepció. Aquells que la permeten solen tenir un límit de nivells d'imbricació, el qual normalment és alt i és difícil arribar-hi. El fet d'arribar-hi cal entendre'l com un senyal d'un mal disseny de l'algorisme, ja que indica que el programa font serà quasi il·legible.

Disseny defectuós d'un algorisme

En aquestes alçades del crèdit no us pot passar això perquè el nivell de complexitat dels programes és ínfim. Més endavant, però, sí que us podria passar en programes més complexos si no utilitzeu tots els recursos que els llenguatges de programació posen a l'abast del programador.

El llenguatge C permet la imbricació, però cal posar-hi molta atenció. Fixeu-vos bé en els exemples següents.

Exemple 1 d'imbricació d'estructures condicionals

Volem fer un programa que demani a l'usuari dos valors numèrics i, posteriorment, informi com és el primer número respecte del segon (major, menor o igual).

Està clar que necessitem dues variables per a emmagatzemar els valors introduïts per l'usuari. Decidim valors reals perquè d'aquesta manera podem treballar amb decimals i sense decimals.

```
programa comparar_valors  és
   var a,b : real;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi dos valors numèrics");
   llegir (a,b);
   si a > b llavors
      escriure ( a, " és major que ", b);
   sinó
      si a < b llavors
         escriure ( a, " es menor que ", b);
      sinó
         escriure ( a, " és igual a ", b);
```

```
      fisi
   fisi
   saltar_línia;
fiprograma
```

Codifiquem-lo en llenguatge C:

```
/* u2n1p03.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   float a,b;
   clrscr();
   printf ("Introdueixi dos valors numèrics: ");
   scanf ("%f %f",&a,&b);
   if (a>b)
      printf ("%g és major que %g\n", a, b);
   else
      if (a<b)
         printf ("%g és menor que %g\n", a, b);
      else
         printf ("%g és igual a %g\n", a, b);
}
```

Trobareu l'arxiu `u2n1p03.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

En aquest cas, la codificació en C de l'algorisme ha estat una simple traducció i el resultat és correcte.

Exemple 2 d'imbricació d'estructures condicionals

Volem fer un programa que demani a l'usuari un caràcter i, posteriorment, informi si el caràcter és una vocal. En cas que ho sigui, i si és una vocal dèbil, es vol que n'informi.

Anàlisi funcional

Cal saber què és una vocal. És clar, oi? Sabem que les vocals són les lletres a, e, i, o i u. Aquí ens hem d'adonar que per a l'ordinador, els caràcters A, E, I, O, U i les corresponents variacions de les majúscules i minúscules puntuades (accents greus, aguts i circumflexos i les dièresis) són caràcters diferents i que per a l'usuari també són vocals.

Per tant, la condició per mirar si és una vocal no consistirà només en cinc comparacions, sinó en cinquanta (10 caràcters entre majúscules i minúscules per a les cinc variacions: sense puntuació, amb accent greu-agut-circumflex, amb dièresi).

És clar que en funció de l'idioma, algunes possibilitats no són factibles, però les podem considerar totes.

Per no escriure la condició amb les cinquanta possibilitats, considerarem només les variants de majúscula i minúscula. Així ja tindreu idea de com caldria acabar la condició per a controlar les cinquanta possibilitats.

Anàlisi orgànica

Arribem a la conclusió que necessitem una variable de tipus caràcter per a recollir el caràcter que introduirà l'usuari.

Plantegem l'algorisme en el pseudocodi següent:

```
programa és_vocal és
   var c : caràcter;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi un caràcter: ");
   llegir (c);
   si c == 'A' o c=='E' o c=='I' o c=='O' o c=='U' o c == 'a' o c=='e'
      o c=='i' o c=='o' o c=='u' llavors
      escriure ( c, " és una vocal ");
      si c=='U' o c=='I' o c=='u' o c=='i' llavors
         escriure (" i és dèbil");
      fisi
   sinó
      escriure ( c, " no és una vocal.");
   fisi
   saltar_línia;
fiprograma
```

O sigui,

- si l'usuari introdueix X, el programa diu X no és una vocal;
- si l'usuari introdueix A, el programa diu A és una vocal;
- si l'usuari introdueix U, el programa diu U és una vocal i és dèbil.

Codifiquem-lo en llenguatge C:

```
/* u2n1p04.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   char c;
   clrscr();
   printf ("Introdueixi un caràcter: ");
   scanf ("%c",&c);
   if (c=='a' || c=='e' || c=='i' || c=='o' || c=='u' || \
       c=='A' || c=='E' || c=='I' || c=='O' || c=='U')
      printf ("%c és una vocal", c);
      if (c=='i' || c=='u'|| c=='I' || c=='U')
         printf (" i és dèbil");
```

Trobareu l'arxiu `u2n1p04.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
   else
      printf ("%c no és una vocal", c);
   printf ("\n");
}
```

L'anterior sembla una codificació correcta, però no ho és. Observem que:

- Si l'usuari introdueix X, el programa diu X no és una vocal. Correcte!
- Si l'usuari introdueix A, el programa diu A és una vocalA no és una vocal. Incorrecte!
- Si l'usuari introdueix U, el programa diu U és una vocal i és dèbil. Correcte!

L'algorisme en pseudocodi és correcte. Com és possible que l'execució del programa en C sigui incorrecta? La resposta és que la codificació en C no és correcta, no reflecteix l'algorisme en pseudocodi.

En pseudocodi, les estructures condicionals `si...llavors... [sinó...]fisi` estan ben definides perquè les paraules reservades ens delimiten exactament el conjunt d'instruccions que cal executar en cada cas. En llenguatge C, però, això no és així, ja que, com que no té la paraula reservada equivalent a `fisi`, quan tenim imbricació d'estructures ens podem trobar un `else` amb ambigüitat respecte a l'`if` al qual correspon. Fixem-nos en l'exemple anterior.

```
if (c=='a' || c=='e' || c=='i' || c=='o' || c=='u' || \
    c=='A' || c=='E' || c=='I' || c=='O' || c=='U') [1]
   printf ("%c és una vocal", c);
   if (c=='i' || c=='u'|| c=='I' || c=='U') [2]
      printf (" i és dèbil");
else [3]
   printf ("%c no és una vocal", c);
```

L'`if` de la línia [2] no té `else`. L'`if` de la línia [1] sí que en té, el de la línia [3]. Però el llenguatge C no té manera de saber l'`else` de la línia [3], a quin `if` pertany, ja que tots dos `if` estan oberts. En pseudocodi era clar que pertanyia al de la línia [1], ja que el de la línia [2] estava tancat amb la paraula reservada `fisi`.

El llenguatge C entén que quan es troba diversos `if` oberts, és a dir, no aparellats amb cap `else`, el primer `else` que es troba l'aparella amb l'`if` més proper no aparellat. Per això, en el cas que ens ocupa, l'`else` de la línia [3] l'aparella amb l'`if` de la línia [2], cosa no desitjada.

La solució, per tant, és utilitzar convenientment les claus, de manera que quedi clar cada `else` a quin `if` pertany. La codificació correcta seria:

```
/* u2n1p05.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{  char c;
   clrscr();
   printf ("Introdueixi un caràcter: ");
   scanf ("%c",&c);
   if (c=='a' || c=='e' || c=='i' || c=='o' || c=='u' || \
       c=='A' || c=='E' || c=='I' || c=='O' || c=='U')
      {  printf ("%c és una vocal", c);
         if (c=='i' || c=='u'|| c=='I' || c=='U')
            printf (" i és dèbil");
      }
   else
      printf ("%c no és una vocal", c);
   printf ("\n");
}
```

Trobareu l'arxiu `u2n1p05.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Exemple 3 d'imbricació d'estructures condicionals

Considerem un darrer exemple d'estructures condicionals imbricades que ens servirà per a veure la necessitat de disposar de l'estructura condicional múltiple.

Volem fer un programa que cerqui el menor de tres valors numèrics introduïts per l'usuari.

Per a fer l'anàlisi orgànica necessitem tres variables numèriques (suposarem que són reals) per a emmagatzemar els valors introduïts per l'usuari. Farem servir una quarta variable també numèrica per a guardar el valor menor en les diferents comprovacions que fem i, quan acabem les comprovacions, informarem del valor emmagatzemat en aquesta quarta variable.

Algorisme en pseudocodi:

```
programa menor_de_tres_valors és
   var a, b, c, menor: real;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi tres valors numèrics: ");
   llegir (a, b, c);
   si a < b llavors
      si a < c llavors menor = a;   sinó  menor = c;  fisi
   sinó
      si b < c llavors menor = b;   sinó menor = c;   fisi
   fisi
   escriure ( "El menor de ", a, ",", b, " i ", c, " és ", menor);
   saltar_línia;
fiprograma
```

Codificació en llenguatge C:

```
/* u2n1p06.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   float a,b,c,menor;
   clrscr();
   printf ("Introdueixi tres valors numèrics: ");
   scanf ("%f %f %f",&a,&b,&c);
   if (a<b)
      if (a<c) menor=a; else menor=c;
   else
      if (b<c) menor=b; else menor=c;
   printf ("El menor de %g, %g i %g és %g\n",a,b,c,menor);
}
```


Trobareu l'arxiu `u2n1p06.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

1.4. Estructura condicional múltiple

Hi ha ocasions en les que és necessari efectuar múltiples imbricacions d'estructures condicionals. Alguns llenguatges incorporen unes estructures condicionals múltiples per a facilitar la claredat i llegibilitat dels algorismes.

L'estructura condicional múltiple permet controlar el fet que en complir-se un cas d'entre un conjunt finit de casos, s'executi el conjunt d'instruccions corresponent.

En general ens podem trobar amb dos tipus d'estructures condicionals múltiples. Alguns llenguatges en suporten les dues, d'altres només una i fins i tot ens podem trobar llenguatges que no en tinguin cap i, en tal cas, cal utilitzar la imbricació d'estructures condicionals.

1.4.1. Estructura condicional `opció` en pseudocodi i ordinogrames

Aquesta estructura condicional és la més potent de les dues perquè permet que cada cas del conjunt finit de casos sigui una condició lògica semblant a la de les estructures condicionals simples i dobles.

La sintaxi en pseudocodi és:

```
opció
   cas <condició_lògica_1>: <conjunt_d'instruccions_1>
   cas <condició_lògica_2>: <conjunt_d'instruccions_2>
   ...
   cas <condició_lògica_n>: <conjunt_d'instruccions_n>
   [altrament: <conjunt_d'instruccions>]
fiopció
```

Quan s'executa una instrucció `opció`, el programa avalua les diferents condicions lògiques fins a trobar-ne una que doni `cert` com a resultat; en aquest moment executa el conjunt d'instruccions que acompanya la condició lògica avaluada com a `certa` i, posteriorment, continua l'execució per la instrucció posterior a la paraula reservada `fiopció`. S'atura a la primera condició lògica avaluada com a `certa`. Si posteriorment n'hi ha d'altres, no se n'assabenta. El cas `altrament` s'executa si no es compleix cap dels `casos` especificats amb les condicions lògiques. Si no es compleix cap dels `casos` i no hi ha `altrament`, s'executa la instrucció següent a la paraula reservada `fiopció`.

Aquesta estructura no té una representació específica en ordinograma, però es pot plantejar com una sèrie d'estructures condicionals dobles imbricades, com mostra la figura 3.

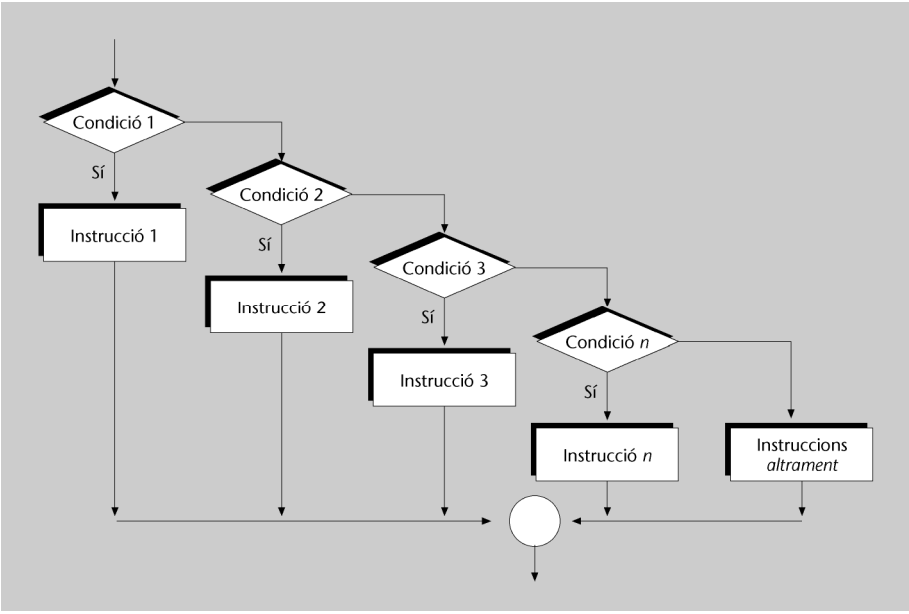


Figura 3 . Representació de l'estructura condicional `opció` en ordinogrames

En l'ordinograma de la figura 3 ha aparegut un nou símbol (rodona inferior) que no havíem fet servir fins ara. Es tracta del connector, utilitzat, tal com el seu nom indica, per a connectar diferents camins en un de sol.

1.4.2. Estructura condicional opció en llenguatge C

El llenguatge C, malauradament, no disposa d'aquesta estructura. Per fer front a aquest inconvenient treballarem amb la imbricació d'estructures condicionals de la manera següent:

```
if (<condició_lògica_1>)
   { <conjunt d'instruccions 1>; }
else if (<condició_lògica_2>)
   { <conjunt d'instruccions 2>; }
else if (<condició_lògica_3>)
   { <conjunt d'instruccions 3>; }
...
[else {<conjunt d'instruccions altrament>;}]
```

L'avaluació d'aquesta estructura succeeix així: si es verifica la condició_lògica_1, s'executa el conjunt_d'instruccions_1 i si no es verifica s'examinen seqüencialment les condicions següents fins al darrer `else`, i s'executa el conjunt d'instruccions corresponent al primer `else if` tal que la seva condició sigui certa. Si totes les condicions són falses, s'executa el conjunt d'instruccions corresponent al darrer `else`. En qualsevol cas es continua en la primera instrucció que hi hagi a continuació de l'estructura.

És a dir, el llenguatge C no té una estructura condicional `opció` específica però és senzilla de construir sobre la base d'`else if`.

Altres llenguatges tenen el mateix handicap i ja incorporen la partícula else if per a aquesta finalitat.

1.4.3. Exemples d'utilització de l'estructura condicional `opció`.

Vegem dos exemples d'utilització de l'estructura condicional `opció`.

Exemple 1 d'utilització de l'estructura condicional `opció`

Considerem el programa que consisteix a cercar el menor valor de tres valors numèrics introduïts per l'usuari.

Algorisme en pseudocodi utilitzant l'estructura condicional `opció`:

```
programa menor_de_tres_valors és
   var a, b, c, menor: real;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi tres valors numèrics: ");
   llegir (a, b, c);
   opció
      cas a ≤ b i a ≤ c : menor = a;
      cas b ≤ a i b ≤ c : menor = b;
      altrament : menor = c;
   fiopció
```

```
   escriure ( "El menor de ", a, ",", b, " i ", c, " és ", menor);
   saltar_línia;
fiprograma
```

Oi que és més entenedor que no pas utilitzar imbricacions d'estructures condicionals simples i dobles?

Codifiquem-lo en llenguatge C utilitzant el muntatge `else if`:

```
/* u2n1p07.C */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   float a,b,c,menor;
   clrscr();
   printf ("Introdueixi tres valors numèrics: ");
   scanf ("%f %f %f",&a,&b,&c);
   if (a<=b && a<=c) menor=a;
   else if (b<=a && b<=c) menor=b;
   else menor=c;
   printf ("El menor de %g, %g i %g és %g\n",a,b,c,menor);
}
```

Trobareu l'arxiu `u2n1p07.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Exemple 2 d'utilització de l'estructura condicional `opció`

Fer un programa que ens ordeni, de menor a major, tres valors numèrics introduïts per l'usuari.

Anàlisi funcional

Per ordenar tres valors numèrics primer hem de trobar el menor o el major i, posteriorment, cercar el menor o major dels dos restants. Per tant, utilitzarem l'esquema de l'exemple anterior (que calcula el menor de tres valors) i l'ampliarem amb el que fa falta per a aquest programa.

Anàlisi orgànica

A més de les tres variables per a emmagatzemar els valors introduïts per l'usuari, definim tres variables més on emmagatzemarem els valors ordenats.

Algorisme en pseudocodi:

```
programa menor_de_tres_valors és
   var a, b, c, menor, mig, major: real;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi tres valors numèrics: ");
```

```
   llegir (a, b, c);
   opció
      cas a ≤ b i a ≤ c :
         menor = a;
         si b ≤ c
            llavors mig=b; major=c;
            sinó mig=c; major=b;
         fisi
      cas b ≤ a i b ≤ c :
         menor = b;
         si a ≤ c
            llavors mig=a; major=c;
            sinó mig=c; major=a;
         fisi
      altrament :
         menor = c;
         si a ≤ b
            llavors mig=a; major=b;
            sinó mig=b; major=a;
         fisi
   fiopció
   escriure ( "Valors ordenats: ",menor,mig,major);
   saltar_línia;
fiprograma
```

Codificació en llenguatge C:

```
/* u2n1p08.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   float a,b,c,menor,mig,major;
   clrscr();
   printf ("Introdueixi tres valors numèrics: ");
   scanf ("%f %f %f",&a,&b,&c);
   if (a<=b && a<=c)
   {  menor=a;
      if (b<c) { mig=b; major=c;}   else { mig=c; major=b;}
   }
   else if (b<=a && b<=c)
         {  menor=b;
            if (a<c) { mig=a; major=c;}   else { mig=c; major=a;}
         }
   else  {  menor=c;
            if (a<b) { mig=a; major=b;}   else { mig=b; major=a;}
         }
   printf ("Valors ordenats són: %g, %g, %g\n", menor, mig, major);
}
```


Trobareu l'arxiu `u2n1p08.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

1.4.4. Estructura condicional `encasde` en pseudocodi i ordinogrames

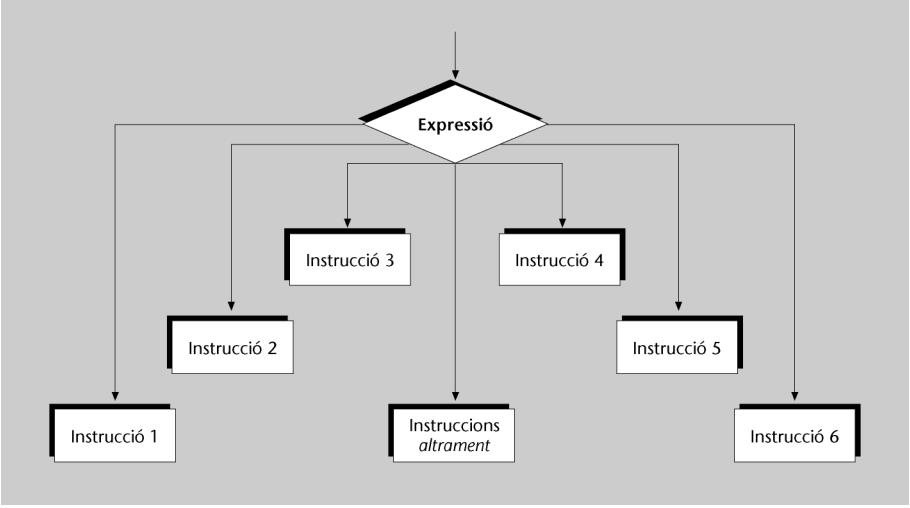
Aquesta estructura condicional permet definir cada cas en funció d'un conjunt de valors en què s'ha de situar el resultat d'una expressió definida a l'inici de l'estructura.
La sintaxi en pseudocodi és:

```
encasde <expressió>
   ser <conjunt_de_valors_1>: <conjunt_d'instruccions_1>
   ser <conjunt_de_valors_2>: <conjunt_d'instruccions_2>
   ...
   ser <conjunt_de_valors_3>: <conjunt_d'instruccions_3>
   [altrament: <conjunt_d'instruccions>]
fiencasde
```

Quan s'executa una instrucció `encasde`, el programa executa l'expressió que dóna un resultat d'algun tipus conegut pel llenguatge; posteriorment, el programa compara aquest resultat amb els diferents conjunts de valors fins a trobar-ne un que contingui el resultat calculat inicialment; en aquest moment, executa el conjunt d'instruccions que acompanya el conjunt de valors i, posteriorment, continua l'execució per la instrucció posterior a la paraula reservada `fiencasde`. S'atura en el primer conjunt de valors que inclou el resultat. Si posteriorment n'hi ha d'altres, no se n'assabenta. El cas `altrament` s'executa si cap conjunt de valors no inclou el resultat calculat. Si cap conjunt de valors no conté el resultat i no hi ha `altrament`, s'executa la instrucció següent a la paraula reservada `fiencasde`.

La figura 4 mostra la representació d'aquesta estructura en ordinograma i consisteix en una generalització de l'estructura seqüencial doble on els camins de sortida poden ser més de dos i de tipus no lògic.

Figura 4. Representació de l'estructura condicional `encasde` en un ordinograma.



Cada camí de sortida correspon a un dels conjunts de valors especificats en els diferents apartats `ser` de l'estructura.
Els conjunts de valors es poden especificar de diferents maneres:

- valor únic;
- llista de valors (separats per comes: 'A', 'E', 'I', 'O', 'U');
- interval de valors (des de l'inici fins al final separats per ..: 'a'..'z').

1.4.5. Estructura condicional `switch` en llenguatge C

El llenguatge C incorpora l'estructura condicional `encasde` per la situació en què l'expressió sigui entera. Estem davant la instrucció `switch` que té la sintaxi següent:

```
switch (<expressió>)
{
   case <expressió_constant_1>: [<conjunt_instruccions_1>]
   case <expressió_constant_2>: [<conjunt_instruccions_2>]
   ...
   case <expressió_constant_n>: [<conjunt_instruccions_n>]
   [default:] [<conjunt_instruccions_altrament>]
}
```

en la qual:

- <expressió> ha de donar un resultat enter; en particular pot ser un caràcter, ja que en C tot caràcter és considerat com un enter (el lloc que ocupa en el codi ASCII);

- <expressió_constant_i> ha de ser una constant entera o una expressió formada per constants amb resultat final enter.

A diferència de la immensa majoria d'estructures condicionals múltiples en llenguatges de programació, la sentència `switch` del llenguatge C té un comportament molt especial. De manera semblant a tots els llenguatges, el programa avalua l'expressió i inicia la comparació amb la constant de cada `case`. En trobar un `case` tal que la constant coincideix amb el resultat, es comença a executar el conjunt d'instruccions corresponent, i continua amb els conjunts d'instruccions dels següents `case` fins a acabar l'estructura `switch`. Només hi ha una manera d'avortar l'execució dels diferents conjunts d'instruccions: la instrucció `break`, que transfereix l'execució a fora del `switch`. Per això té sentit que en alguns casos el conjunt d'instruccions sigui optatiu, ja que s'executarà el(s) conjunt(s) d'instruccions del(s) `case` següent(s).
L'estructura `switch` del llenguatge C és més potent que l'estructura `encasde` ja que dóna més joc al programador. En cas que la vulguem fer

servir amb la mateixa funcionalitat només cal utilitzar la instrucció `break` al final de cada bloc d'instruccions, tal com es veu a continuació:

```
switch (<expressió>)
{
   case <expressió_constant_1>:  <conjunt_instruccions_1>
                                       break;
   case <expressió_constant_2>:  <conjunt_instruccions_2>
                                       break;
   ...
   case <expressió_constant_n>:  <conjunt_instruccions_n>
                                       break;
   default: <conjunt_instruccions_altrament>
}
```

1.4.6. Exemples d'utilització de l'estructura `encasde`.

Vegem dos exemples d'utilització de l'estructura condicional `encasde` comparant amb la possible utilització de l'estructura condicional `opció`.

Exemple 1 d'utilització de l'estructura condicional `encasde`

Volem fer un programa que faciliti la traducció del mes numèric a mes en lletres (1 per a gener, 2 per a febrer, etc.)

Anàlisi funcional

Per a resoldre el problema necessitem saber els noms dels dotze mesos. No hi ha més complicació.

Anàlisi orgànica

Considerarem una variable de tipus `natural` per a rebre el valor numèric del mes introduït per l'usuari. Podem fer dues versions utilitzant l'estructura condicional `opció` i l'estructura condicional `encasde`.

Versió en pseudocodi utilitzant l'estructura concidional `opció`:

```
programa traducció_mes_1 és
   var mes: natural;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi el número de mes a traduir, si li plau:");
   llegir (mes);
   saltar_línia;
   opció
      cas mes==1: escriure ("Traducció: Gener");
      cas mes==2: escriure ("Traducció: Febrer");
      cas mes==3: escriure ("Traducció: Març");
```

```
      cas mes==4: escriure ("Traducció: Abril");
      cas mes==5: escriure ("Traducció: Maig");
      cas mes==6: escriure ("Traducció: Juny");
      cas mes==7: escriure ("Traducció: Juliol");
      cas mes==8: escriure ("Traducció: Agost");
      cas mes==9: escriure ("Traducció: Setembre");
      cas mes==10: escriure ("Traducció: Octubre");
      cas mes==11: escriure ("Traducció: Novembre");
      cas mes==12: escriure ("Traducció: Desembre");
      altrament: escriure ("No existeix un mes amb número ",mes);
   fiopció
   saltar_línia;
fiprograma
```

Versió en pseudocodi utilitzant l'estructura condicional `encasde`:

```
programa traducció_mes_2 és
   var mes: natural;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi el número de mes a traduir, si li plau:");
   llegir (mes);
   saltar_línia;
   encasde mes
      ser 1: escriure ("Traducció: Gener");
      ser 2: escriure ("Traducció: Febrer");
      ser 3: escriure ("Traducció: Març");
      ser 4: escriure ("Traducció: Abril");
      ser 5: escriure ("Traducció: Maig");
      ser 6: escriure ("Traducció: Juny");
      ser 7: escriure ("Traducció: Juliol");
      ser 8: escriure ("Traducció: Agost");
      ser 9: escriure ("Traducció: Setembre");
      ser 10: escriure ("Traducció: Octubre");
      ser 11: escriure ("Traducció: Novembre");
      ser 12: escriure ("Traducció: Desembre");
      altrament: escriure ("No existeix un mes amb número ",mes);
   fiencasde
   saltar_línia;
fiprograma
```

En aquest cas es pot veure que no hi ha gran diferència entre fer servir la forma `opció` i la forma `encasde`.

Ara fem la codificació en llenguatge C de la versió corresponent a la utilització, en pseudocodi, de l'estructura condicional `opció`, amb la utilització de la imbricació d'instruccions `if`:

```
/* u2n1p09.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
```

Trobareu l'arxiu `u2n1p09.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
   unsigned int mes;
   clrscr();
   printf ("Introdueixi el número de mes a traduir: ");
   scanf ("%u",&mes);
   if (mes==1) printf ("Traducció: Gener\n");
   else if (mes==2) printf ("Traducció: Febrer\n");
   else if (mes==3) printf ("Traducció: Març\n");
   else if (mes==4) printf ("Traducció: Abril\n");
   else if (mes==5) printf ("Traducció: Maig\n");
   else if (mes==6) printf ("Traducció: Juny\n");
   else if (mes==7) printf ("Traducció: Juliol\n");
   else if (mes==8) printf ("Traducció: Agost\n");
   else if (mes==9) printf ("Traducció: Setembre\n");
   else if (mes==10) printf ("Traducció: Octubre\n");
   else if (mes==11) printf ("Traducció: Novembre\n");
   else if (mes==12) printf ("Traducció: Desembre\n");
   else printf ("No existeix un mes amb número %u\n",mes);
}
```

Codificació en llenguatge C de la versió corresponent a la utilització, en pseudocodi, de l'estructura condicional `encasde`, amb la utilització de la instrucció `switch`:

```
/* u2n1p10.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int mes;
   clrscr();
   printf ("Introdueixi el número de mes a traduir: ");
   scanf ("%u",&mes);
   switch (mes)
   {
      case 1: printf ("Traducció: Gener\n"); break;
      case 2: printf ("Traducció: Febrer\n"); break;
      case 3: printf ("Traducció: Març\n"); break;
      case 4: printf ("Traducció: Abril\n"); break;
      case 5: printf ("Traducció: Maig\n"); break;
      case 6: printf ("Traducció: Juny\n"); break;
      case 7: printf ("Traducció: Juliol\n"); break;
      case 8: printf ("Traducció: Agost\n"); break;
      case 9: printf ("Traducció: Setembre\n"); break;
      case 10: printf ("Traducció: Octubre\n"); break;
      case 11: printf ("Traducció: Novembre\n"); break;
      case 12: printf ("Traducció: Desembre\n"); break;
      default: printf ("No existeix un mes amb número %u\n",mes);
   }
}
```

Trobareu l'arxiu `u2n1p10.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

En aquest exemple s'ha pogut comprovar que no hi ha gaires avantatges a l'hora de fer servir una estructura o l'altra. Hi ha, però, casos en que la instrucció `switch` del llenguatge C facilita molta potència. (que ni tan sols té la instrucció `encasde` del pseudocodi).

Exemple 2 d'utilització de l'estructura condicional `encasde`

Volem un programa que ens digui quants dies té un mes introduït per l'usuari.

Anàlisi funcional

Sabem que excepte el mes de febrer, tots els mesos tenen un determinat nombre de dies independentment de l'any. En canvi, el mes de febrer depèn de l'any, ja que si és un any de traspàs té 29 dies i en cas contrari en té 28.

I quan un any és de traspàs? El coneixement popular diu que ens trobem un any de traspàs cada quatre anys, coincidint amb l'any que és divisible per quatre, és a dir: 1992, 1996, 2000, etc. No és exactament així. Un any és de traspàs quan es verifica alguna de les dues condicions següents:

Aquest és un exemple clar del que ha de fer l'analista: esbrinar la realitat, la qual pot estar amagada.

- és múltiple de 4 i no ho és de 100;
- és múltiple de 400.

Així, l'any 2000 no seria de traspàs per la primera condició (és multiple de 4 i també ho és de 100), però sí que en compleix la segona (és múltiple de 400).

Exemple d'any suposadament de traspàs

L'any 3000, per exemple, no serà de traspàs, i en canvi la majoria de la gent pensa que sí que ho és. De fet, suposem que això no us preocupa gaire, oi?

Per tant, per a poder resoldre el problema, caldrà que l'usuari, a més d'introduir el mes, introdueixi l'any. De fet, la segona dada només es necessitarà en el cas que el mes introduït correspongui a febrer.

Anàlisi orgànica

Necessitarem dues variables i aplicarem els càlculs corresponents a partir de l'anàlisi funcional.

Versió en pseudocodi utilitzant l'estructura condicional `opció`:

```
programa dies_del_mes_1 és
   var mes, any: natural;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi el número de mes, si li plau:");
   llegir (mes);
   saltar_línia;
   opció
      cas mes==1 o mes==3 o mes==5 o mes==7 o mes==8 o  mes==10 o mes==12 :
         escriure ("Té 31 dies");
```

```
      cas mes==4 o mes==6 o mes==9 o mes==11 :
         escriure ("Té 30 dies");
      cas mes==2 :
         escriure ("Mes de febrer. Introdueixi l'any:");
         llegir (any);
         si (any mod 4 == 0 i any mod 100 != 0) o any mod 400 == 0
            llavors escriure ("Té 29 dies");
            sinó escriure ("Té 28 dies");
         fisi
      altrament: escriure ("No existeix un mes amb número ",mes);
   fiopció
   saltar_línia;
fiprograma
```

Vegem com es pot simplificar utilitzant l'estructura encasde:

```
programa dies_del_mes_2 és
   var mes, any: natural;
   fivar
   netejar_pantalla;
   escriure ("Introdueixi el número de mes, si li plau:");
   llegir (mes);
   saltar_línia;
   encasde mes
      ser 1,3,5,7,8,10,12 :  escriure ("Té 31 dies");
      ser 4,6,9,11 : escriure ("Té 30 dies");
      ser 2 :
         escriure ("Mes de febrer. Introdueixi l'any:");
         llegir (any);
         si (any mod 4 == 0 i any mod 100 != 0) o any mod 400 == 0
            llavors escriure ("Té 29 dies");
            sinó escriure ("Té 28 dies");
         fisi
      altrament: escriure ("No existeix un mes amb número ",mes);
   fiencasde
   saltar_línia;
fiprograma
```

Codificació en llenguatge C de la versió corresponent a la utilització, en pseudocodi, de l'estructura condicional `opció`, amb la utilització de la imbricació d'instruccions `if`:

```
/* u2n1p11.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int mes, any;
   clrscr();
   printf ("Introdueixi el número de mes: ");
   scanf ("%u",&mes);
   if (mes==1 || mes==3 || mes ==5 || mes==7 || mes==8 || mes==10 || \
       mes==12)
      printf ("Té 31 dies\n");
```


Trobareu l'arxiu `u2n1p11.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
   else if (mes==4 || mes==6 || mes==9 || mes==11)
      printf ("Té 30 dies\n");
   else if (mes==2)
         {  printf ("Mes de febrer. Introdueixi l'any: ");
            scanf ("%u", &any);
            if ((any % 4 == 0 && any % 100 != 0) ||any % 400 == 0)
               printf ("Té 29 dies\n");
            else printf ("Té 28 dies\n");
         }
   else printf ("No existeix un mes amb número %u\n",mes);
}
```

Codificació en llenguatge C de la versió corresponent a la utilització, en pseudocodi, de l'estructura condicional `encasde`, amb la utilització de la instrucció `switch`:

```
/* u2n1p12.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int mes, any;
   clrscr();
   printf ("Introdueixi el número de mes: ");
   scanf ("%u",&mes);
   switch (mes)
   {  case 1:  case 3:  case 5:  case 7:  case 8:  case 10: case 12:
         printf ("Té 31 dies\n"); break;
      case 4:  case 6:  case 9:  case 11:
         printf ("Té 30 dies\n"); break;
      case 2:  printf ("Febrer! Introdueixi l'any: ");
               scanf ("%u", &any);
               if ((any % 4 == 0 && any % 100 != 0) ||any % 400 == 0)
                     printf ("Té 29 dies\n");
               else printf ("Té 28 dies\n");
               break;
      default: printf ("No existeix un mes amb número \ %u\n", mes);
   }
}
```

Trobareu l'arxiu `u2n1p12.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

En aquest segon exemple es pot veure com se simula una llista de valors gràcies a l'execució de les instruccions dels següents `case` fins a trobar una instrucció `break`.

Per altra banda, també heu d'observar com s'ha codificat les condicions corresponents a la confirmació de l'any de traspàs:

```
(any mod 4 == 0 i any mod 100 != 0) o any mod 400 == 0
```

La traducció literal de l'expressió anterior en C comporta:

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

Recordeu, però, que en llenguatge C un valor enter no zero és considerat com a `cert` i un valor enter zero és considerat com a `fals`. Per tant, l'anterior expressió lògica pot ser escrita en C de la maner següent:

```
((!(any % 4) && any % 100) || !(any % 400))
```

ja que:

- Si `any % 4 == 0` és cert, vol dir que `any % 4` és `zero`, o sigui, `fals`, i per tant, si volem aconseguir que sigui `cert` només cal afegir la negació al seu davant; per tant, l'expressió inicial és equivalent a `!(any % 4)`. De manera semblant passa amb `any % 400 == 0`, que és equivalent a `!(any % 400)`.

- Si `any % 100 != 0` és `cert`, vol dir que `any % 100` no és `zero`, o sigui, `cert`, i per tant, l'expressió es pot substituir directament per `any % 100`.

2. Estructures repetitives

El segon grup de sentències de control que estudiarem són les estructures repetitives.

> Les estructures repetitives o iteratives permeten repetir una mateixa seqüència d'instruccions fins que es compleixi o es deixi de complir una sèrie de condicions.

De manera semblant a les estructures condicionals, l'estructura repetitiva té, en algun punt, l'avaluació d'una determinada expressió que permet decidir la continuació del procés o el seu avortament. És a dir, existeix també una condició lògica.

> Anomenem bucle o cicle el conjunt d'instruccions que s'ha de repetir un nombre determinat de vegades.
>
> Anomenem iteració cada execució del bucle.

2.1. Variables de control de les estructures repetitives: semàfors, comptadors, acumuladors

En les estructures condicionals hem comentat que no tenia sentit que la condició lògica no pogués prendre qualsevol dels dos valors. Hem dit que no tenia sentit posar una condició lògica tal que en sapiguem el resultat en temps d'edició del programa font i que no depengui del moment d'execució.

De manera semblant, en les estructures repetitives la condició lògica mantindrà un determinat valor que permeti anar repetint el bucle, però caldrà tenir la seguretat que arribarà un moment en què la condició lògica canviï de valor per tal d'assegurar la sortida del bucle. En cas contrari, entraríem en el que s'anomena un bucle infinit, la qual cosa provoca la "penjada" del programa, ja que no evoluciona i, evidentment, no acaba. (!!)

I com pot canviar el valor de la condició lògica? Doncs sobre la base dels valors de les variables que es van modificant dins el mateix bucle.

És a dir, forçosament hi ha variables que n'han d'anar modificant el valor i que ens permeten controlar la repetició o el final del bucle: són variables de control.

Aquestes variables ens poden servir, a més, per a altres finalitats. Totes les hem de declarar en el programa i poden ser de diferents tipus de dades.

Semàfors

Són variables que només poden prendre dos valors: 0-1, cert-fals, sí-no, etc. També s'anomenen interruptors, banderes, flags o switchs.

S'acostumen a utilitzar per a repetir el bucle fins que canvien el seu valor; és a dir, es fan servir per a provocar la fi de l'execució del bucle.

Comptadors

Són variables de tipus enter, el valor de les quals augmenta o disminueix d'una manera constant a cada execució del bucle.

S'acostumen a utilitzar per a comptar quantes vegades s'efectua una determinada operació (o conjunt d'operacions). A vegades ens serveixen per a acabar l'execució del bucle quan s'ha arribat a un cert nombre de vegades.

Acumuladors

Són variables que acumulen quantitats variables per sumes o restes successives. També s'anomenen sumadors o totalitzadors.

Es poden fer servir per a acabar l'execució del bucle quan s'ha arribat a una certa quantitat acumulada.

2.2. Estructura repetitiva `mentre...fer`

La primera estructura repetitiva que estudiarem s'anomena `mentre...fer` i té a veure amb les condicions lògiques que hem vist en l'apartat anterior.

L'estructura repetitiva `mentre...fer` permet repetir l'execució del bucle mentre es verifiqui la condició lògica. Abans d'iniciar la primera iteració, la condició ja s'ha de verificar. En cas contrari, no s'executa cap iteració.

Igual que en l'estudi de les estructures condicionals, veurem com s'expressa l'estructura interactiva `mentre...fer` en pseudocodi, ordinogrames i llenguatge C.

2.2.1. Estructura `mentre...fer` en pseudocodi i ordinogrames

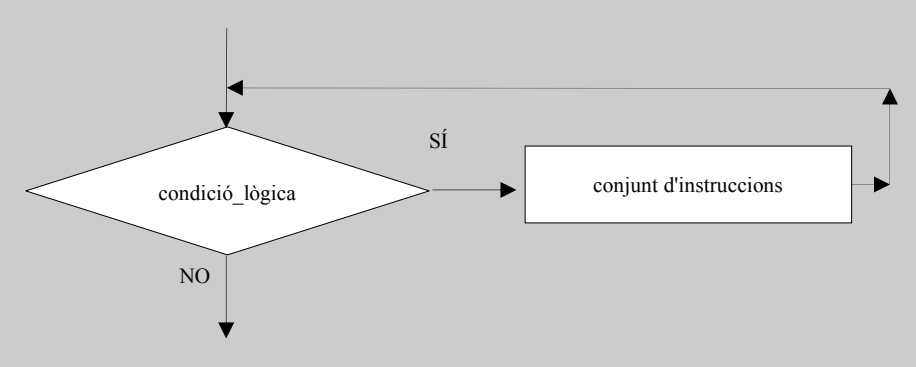
La sintaxi en pseudocodi és la següent:

```
mentre <condició_lògica> fer
   <conjunt d'instruccions>
fimentre
```

El programa, en temps d'execució, avalua la condició lògica. Si és avaluada com a certa, s'executa el conjunt d'instruccions i es torna a repetir el procés d'avaluació de la condició lògica. Quan és avaluada com a falsa, el programa continua l'execució a la instrucció que hi ha després de la paraula reservada `fimentre`.

La figura 5 mostra la representació de l'estructura repetitiva `mentre...fer` en un ordinograma.

Figura 5. Representació de l'estructura repetitiva `mentre...fer` en un ordinograma



2.2.2. Estructura `while` en llenguatge C

El llenguatge C també disposa de l'estructura repetitiva `mentre...fer`. És l'estructura `while` i la seva sintaxi és la següent:

```
while ([<condició_lògica>])
{  <conjunt d'instruccions a executar>; }
```

Heu de tenir en compte els punts següents:

- És obligatori emmarcar la condició lògica entre parèntesis.
- No hi ha paraules reservades equivalents a les paraules `fer` i `fimentre` del pseudocodi. Els símbols {} emmarquen les instruccions tingudes en compte entre el `fer` i el `fimentre`.
- Si el conjunt d'instruccions que hem d'executar consisteix en una única instrucció, els símbols {} es poden estalviar. Per tant, si després de la condició lògica no hi ha text emmarcat entre {}, vol dir que el cos d'instruccions que s'ha d'executar està format per una única instrucció.
- Recordeu que en llenguatge C no existeix el valor `lògic` i, per tant, la condició lògica pot ser una expressió de qualsevol tipus que serà avaluada com a `certa` si no és zero.
- Si no hi ha condició lògica el bucle es va repetint indefinidament. Com es pot interrompre la repetició? Veurem més endavant, en aquesta mateixa unitat didàctica, maneres de trencar l'estructura repetitiva.

2.2.3. Exemples d'utilització de l'estructure `mentre...fer`

Vegem tres exemples d'utilització de l'estructura repetitiva `mentre ... fer` amb grau de dificultat creixent.

Exemple 1 d'utilització de l'estructura condicional `mentre...fer`

Volem fer un programa que demani un text a l'usuari i en compti el nombre de caràcters que conté.

Anàlisi funcional

Hem de tenir alguna manera de saber que l'usuari ha acabat d'introduir el text. Suposarem un símbol no corrent en els textos, el símbol '$'. Per tant, quan el programa detecti un símbol '$' acabarà la comptabilització de caràcters. Cal recordar que l'espai també és un caràcter.

Anàlisi orgànica

Observeu que el programa no ha de memoritzar el text introduït per l'usuari. Únicament ha de comptabilitzar quants caràcters conté. És la mateixa situació que es produeix si algú us demana que compteu quants caràcters té El Quixot. Oi que per fer això no l'heu de memoritzar?

Si ho féssim nosaltres, de manera intuïtiva, quasi sense pensar-ho, partiríem de zero caràcters i per cada caràcter incrementaríem repetidament en una unitat el valor anteriorment emmagatzemat. El procés repetitiu s'acabaria quan comprovés l'arribada del caràcter especial $. Observeu que tenim una variable comptador en el nostre cervell.

Per tant, declararem una variable que tindrà la funció de comptador. A més necessitarem una variable per a poder llegir el caràcter i comprovar quan es produeix l'arribada del caràcter especial.

Algorisme en pseudocodi:

```
programa comptar_caràcters és
   var   c: caràcter;
         n: natural;
   fivar
   escriure ("Introdueixi un text.");
   saltar_línia;
   escriure ("L'aparició del símbol $ indicarà l'acabament.");
   saltar_línia;
   n = 0; /* Inicialitzem l'acumulador */
   llegir (c);
   mentre c != '$' fer
      n = n+1;
      llegir (c);
   fimentre
   escriure ("El text conté ", n, " caràcters abans del símbol $");
   saltar_línia;
fiprograma
```

Codificació en llenguatge C:

```
/* u2n2p01.C */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int n=0;
   char c;
   clrscr();
   printf ("Introdueixi un text.\n");
   printf ("L'aparició del símbol $ indicarà la fi.\n");
```

Trobareu l'arxiu `u2n2p01.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

```
   scanf("%c",&c);
   while (c != '$')
   {  n = n + 1;
      scanf ("%c",&c);
   }
   printf ("El text conté %u caràcters abans de $.\n",n);
}
```

Hem aprofitat el moment de la declaració de la variable `n` per a procedir a la seva inicialització amb el valor zero.

Exemple 2 d'utilització de l'estructura condicional `mentre...fer`

Volem fer un programa que demani un text a l'usuari i en compti el nombre de lletres A que conté.

Anàlisi funcional

Decidim el símbol $ com la marca de final de text.

Anàlisi orgànica

En aquest cas, també caldrà declarar una variable que ens permeti anar acumulant les vegades que apareix la lletra A. No és una variable del tipus comptador vista en l'exemple anterior. Una variable és comptador si s'incrementa o decrementa una quantitat constant a cada iteració. En aquest cas, no es compta a cada caràcter tractat, sinó únicament quan apareix un caràcter A. Es tracta d'un acumulador que ens serveix per a comptar un determinat fet.

Tal com ja s'ha vist anteriorment en algun exemple, els caràcters majúscula i minúscula són diferents, així com els puntuats en les seves diverses versions. En l'algorisme que ens ocupa estem tractant únicament la lletra A.

A més de la variable acumulador, necessitarem una variable de tipus caràcter per a guardar-hi momentàniament el caràcter que s'ha de tractar.

Algorisme en pseudocodi:

```
programa comptar_caràcters és
   var   c: caràcter;
         n: natural;
   fivar
   escriure ("Introdueixi un text.");
   saltar_línia;
   escriure ("L'aparició del símbol $ indicarà l'acabament.");
   saltar_línia;
```

```
   n = 0; /* Inicialitzem l'acumulador */
   llegir (c);
   mentre c != '$' fer
      si  c == 'A'  llavors n = n+1; fisi
      llegir (c);
   fimentre
   escriure ("El text conté ", n, " lletres A abans del símbol $");
   saltar_línia;
fiprograma
```

Codificació en llenguatge C:

```
/* u2n2p02.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int n=0;
   char c;
   clrscr();
   printf ("Introdueixi un text.\n");
   printf ("L'aparició del símbol $ indicarà la fi.\n");
   scanf("%c",&c);
   while (c != '$')
   {  if (c == 'A') n++;
      scanf ("%c",&c);
   }
   printf ("El text conté %u lletres A abans de $.\n",n);
}
```


Trobareu l'arxiu `u2n2p02.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

A diferència de l'exemple anterior, en el qual a cada iteració calia incrementar la variable `n`, en aquest exemple només cal incrementar-la en confirmar que es tracta d'una lletra A.

Exemple 3 d'utilització de l'estructura condicional `mentre...fer`

Volem fer un programa que llisti els nombres naturals des d'1 fins a 100 en files que continguin 10 elements.

Anàlisi funcional

Només necessitem saber comptar d'un en un, i ja en sabem, oi?

Anàlisi orgànica

Necessitem una variable per a comptar els diferents valors des d'1 fins a 100. Quan fem l'algorisme en pseudocodi no ens preocuparem que els valors quedin encolumnats. En la posterior codificació en el llenguatge C, sí que ho tindrem present.

Per controlar que cada 10 números es passi a una nova fila, només cal tenir present que l'operació residu de la divisió entera per 10 ens permet controlar-ho.

Algorisme en pseudocodi:

```
programa comptar_fins_a_100 és
   var n: natural;
   fivar
   netejar_pantalla;
   n = 1;
   mentre ( n ≤ 100 ) fer
      escriure (n);
      si n mod 10 == 0 llavors saltar_línia; fisi
      n = n + 1;
   fimentre
fiprograma
```

Codificació en llenguatge C:

```
/* u2n2p03.C */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int n=1;
   clrscr();
   while (n <= 100)
   {  printf ("%5u",n);
      if (n % 10 == 0) printf ("\n");
      n++;
   }
}
```

Trobareu l'arxiu `u2n2p03.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Tingueu en compte el següent:

- Per a mantenir encolumnats tots els valors utilitzem l'apartat amplada de l'especificació de format `%u` de la instrucció `printf`.

- D'altra banda, noteu també que la condició:

  ```
  n % 10 == 0
  ```

  l'hauríem pogut escriure com a:

  ```
  !(n % 10)
  ```

2.3. Estructura repetitiva `repetir...fins`

Un altre tipus d'estructura repetitiva és l'anomenada `repetir...fins`, que estudiarem en aquest apartat.

> L'estructura repetitiva `repetir...fins` permet repetir l'execució del bucle fins que es verifiqui la condició lògica. La primera iteració sempre s'executa.

Vegem com expressar l'estructura repetitiva `repetir...fins` en pseudocodi, ordinogrames i llenguatge C.

2.3.1. Estructura `repetir...fins` en pseudocodi i ordinogrames

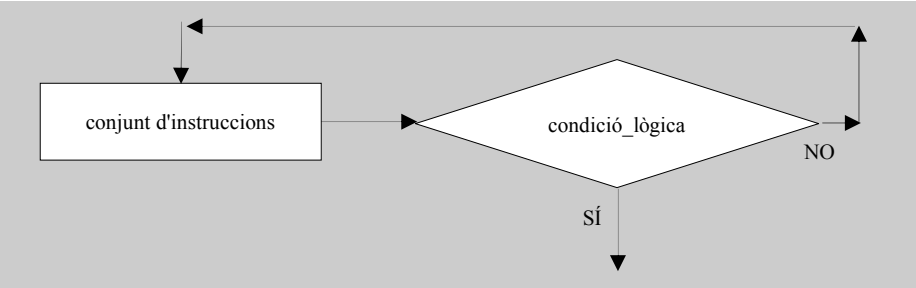
La sintaxi en pseudocodi és la següent:

```
repetir
   <conjunt d'instruccions>
fins <condició_lògica>
```

El programa executa el conjunt d'instruccions fins que es compleix la condició lògica, és a dir, mentre no es compleixi la condició lògica, el bucle s'anirà executant. Sempre s'executa el bucle una primera vegada. Quan la condició lògica és avaluada com a certa, el programa continua l'execució a la instrucció posterior a l'estructura.

La figura 6 mostra la representació de l'estructura repetitiva `repetir...fins` en un ordinograma.

Figura 6. Representació de l'estructura repetitva `repetir...fins` en un ordinograma.



Variant `fer...mentre`

La característica primordial de l'estructura `repetir...fins` és que assegura l'execució d'una iteració com a mínim. Fixeu-vos que

posteriorment es repetirà el procés fins que es compleixi una certa condició. És a dir, l'avaluació positiva de la condició provoca el final de l'execució del bucle.

Hi ha una variant en la qual el procés es repeteix mentre es compleixi una certa condició. És a dir, l'avaluació negativa de la condició provoca el final de l'execució del bucle.

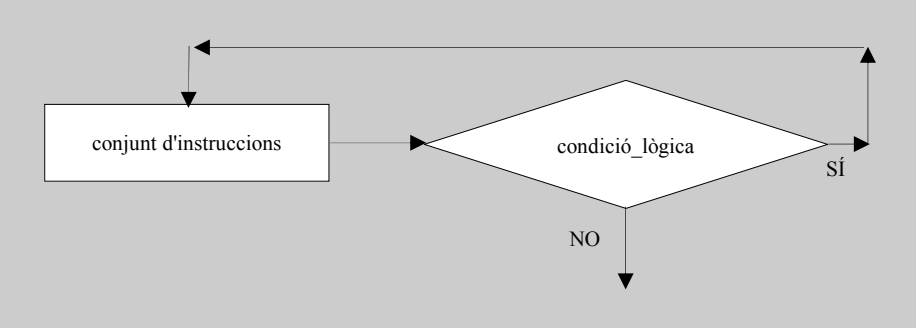
La sintaxi en pseudocodi és:

```
fer
   <conjunt d'instruccions>
mentre <condició_lògica>
```

En aquest cas, també està assegurada l'execució d'una iteració com a mínim.

La figura 7 mostra la representació de l'estructura repetitiva `fer...mentre` en un ordinograma.

Figura 7. Representació de l'estructura repetitiva `fer...mentre` en un ordinograma.



2.3.2. Estructura `do...while` en llenguatge C

El llenguatge C disposa de l'estructura repetitiva equivalent a `fer...mentre`. És l'estructura `do...while` i la seva sintaxi és:

```
do
{  <conjunt d'instruccions a executar>; }
while ([<condició_lògica>])
```

En aquest cas cal tenir en compte les mateixes observacions que havíem notat en l'estructura `while`.

2.3.3. Exemples d'utilització de l'estructura `repetir...fins`

Vegem diferents situacions en les que podem utilitzar l'estructura repetitiva `fer...mentre`.

Exemple 1 d'utilització de l'estructura repetitiva `fer...mentre`.

Reconsiderem els exercicis `u2n2p01`, `u2n2p02` i `u2n2p03` en els que hem fet servir l'estructura `mentre...fer` en aquells casos en què és convenient utilitzar la nova estructura.

És aplicable a l'exemple `u2n2p01`?

Calia fer un programa que demanés un text a l'usuari i comptés el nombre de caràcters que contenia.

Les anàlisis funcional i orgànica no canvien. Recordem que llegíem un primer caràcter i l'analitzàvem per detectar si corresponia a la marca de final de text $. En cas que no ho fos el comptàvem.

En aquest cas no ens serveix la nova estructura, ja que no és segur que el primer caràcter llegit s'hagi de comptar, perquè podria ser directament el $. Per tant, la nova estructura no ens ajuda a millorar el flux d'execució del programa.

És aplicable a l'exemple `u2n2p02`?

Calia fer un programa que demanés un text a l'usuari i comptés el nombre de lletres A que contenia.

Anàlogament al cas anterior, les anàlisis funcional i orgànica no canvien. I de manera també anàloga, no té sentit fer servir la nova estructura, ja que abans d'analitzar la conveniència de comptar o no el primer caràcter (contingut del bucle) cal saber si es tracta de la marca corresponent al final del text.

És aplicable a l'exemple `u2n2p03`?

Calia fer un programa que llistés els nombres naturals des d'1 fins a 100 en files de 10 elements.

Aquí pot tenir sentit utilitzar la nova estructura, ja que si comencem amb el número 1 ja sabem que és menor o igual a 100 i, per tant, segur que cal entrar dins el bucle, per la qual cosa ens podem estalviar la pregunta corresponent.

Algorisme en pseudocodi:

```
programa comptar_fins_a_100 és
   var n: natural;
   fivar
   netejar_pantalla;
   n = 1;
   repetir                        /* fer */
      escriure (n);
      si n mod 10 == 0 llavors saltar_línia; fisi
      n = n + 1;
   fins n>100;          /* mentre n≤100 */
fiprograma
```

Codificació en llenguatge C:

```
/* u2n2p04.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int n=1;
   clrscr();
   do
   {  printf ("%5u",n);
      if (n % 10 == 0) printf ("\n");
      n++;
   } while (n<=100);
}
```

Trobareu l'arxiu `u2n2p04.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Exemple 2 d'utilització de l'estructura repetitiva `repetir...fins`.

En moltes ocasions ens trobarem en la situació d'haver de verificar els valors introduïts per l'usuari en una instrucció d'entrada i haurem d'imposar la repetició de la instrucció d'entrada si els valors introduïts no són verificats.

A l'apartat "Instruccions d'entrada" de la unitat didàctica "Introducció a la programació" hem iniciat l'estudi de les instruccions d'entrada.

Aquesta situació implica la utilització d'una estructura repetitiva que s'acabi quan es verifiquin els requeriments. Considerem un cas concret: suposem que exigim a l'usuari que introdueixi el sexe d'una persona i volem exigir que el valor introduït sigui una D o una H (dona-home).

Considerem l'algorisme en pseudocodi:

```
programa introduir_sexe és
   var sexe: caràcter; fivar
   escriure ("Introdueixi el sexe (D/H)");
   llegir (sexe);
   mentre sexe != 'D' i sexe != 'H' fer llegir (sexe); fimentre
fiprograma
```

És clar que la primera lectura cal efectuar-la sempre. Per tant, podríem utilitzar l'estructura `repetir...fins` o `fer...mentre`. És a dir:

```
programa introduir_sexe és
   var s: caràcter; fivar
   escriure ("Introdueixi el sexe (D/H)");
   fer                              /* repetir */
      llegir (s);
   mentre s != 'D' i s != 'H'       /* fins s == 'D' o s == 'H' */
fiprograma
```

En pseudocodi podem suposar que l'usuari introdueix un valor de tipus corresponent a la variable que està esperant el valor, però en molts llenguatges és responsabilitat del programador efectuar-ne el control. El llenguatge C n'és un. Com s'ha de fer?

Es tracta de controlar el tipus del valor introduït per l'usuari com un requeriment més. La funció `scanf` del llenguatge C ens permetia llegir valors des de l'entrada estàndard (normalment el teclat) i ens informava de quants valors havia pogut llegir (els no llegits quedaven dins el *buffer*). En la codificació de l'exemple s'ha de controlar que l'entrada s'efectuï correctament.

```
/* u2n2p05.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   char sexe;
   unsigned int q;
   clrscr();
   printf ("Introdueixi el sexe (D/H): ");
   do
   q = scanf ("%c", &sexe);
   while (q==0 || (sexe!='H' && sexe!='D'));
   printf ("Sexe introduït: %c\n",sexe);
}
```

Trobareu l'arxiu `u2n2p05.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

La funció `scanf` es queda aturada esperant l'entrada d'un caràcter. El primer caràcter de qualsevol text teclejat per l'usuari és ficat dins la variable `sexe`. Si l'usuari premés directament el return, no s'hauria efectuat cap lectura i podria ser que, per casualitat, el valor emmagatzemat dins la variable `sexe` fos `D` o `H`, la qual cosa provocaria la continuació del programa suposant que veritablement s'hi havia entrat un valor. Fixeu-vos com es controla aquest fet amb la utilització de la variable `q` que conté tots els valors que ha llegit la funció `scanf`. Si `q` és zero indica que no s'ha efectuat tal lectura. En aquest cas, juntament amb el cas que el valor de `sexe` no fos ni `D` ni `H`, s'obliga a tornar a efectuar la lectura.

Encara hi ha un problema. Suposem que davant el programa anterior, l'usuari introdueix la paraula `MARGARIDA`**.** Com actua el programa? Es pot esperar que erròniament. En efecte: l'usuari ha introduït diversos caràcters però la funció `scanf` només en llegeix el primer, `M`, (la resta es queda en el *buffer*). La variable `q` conté el valor `1` però la variable `sexe` conté una `M`. Per tant es torna a executar la funció `scanf`. Però com que en el *buffer* hi troba material (`ARGARIDA`), no provoca l'aturada del programa esperant que l'usuari introdueixi el valor, sinó que agafa el primer caràcter `A` i així es va repetint el procés. Ara ja veieu el que passa, no? Arribarà a la `D` i l'agafarà com a sexe correcte. Fins i tot encara quedarà una `A` dins el *buffer*, que no servirà per a res.

Aquesta situació se soluciona, en algunes plataformes, amb la utilització d'una nova instrucció del llenguatge C: la funció `fflush`, que provoca la neteja del *buffer*. De moment creieu-vos la utilització d'aquesta funció de la manera següent:

```
fflush (stdin);
```

La funció `fflush`

Aprofitem alguns exemples per introduir la utilització en llenguatge C de certes instruccions (en realitat funcions, com ja veurem) que són imprescindibles per a qualsevol programador de C.

La sintaxi `fflush (stdin)` provoca la neteja de les dades pendents de tractar en el *buffer* de l'entrada estàndard (`stdin`). Per tant, podeu intuir que la funció `fflush` té més funcionalitats ja que es podrà fer servir per al tractament de dades pendents en diferents *buffers*. Ho veureu més endavant.

Per tant, la correcta codificació de l'exemple en llenguatge C sembla que podria ser:

```
/* u2n2p06.c */

#include <stdio.h>
#include <conio.h>

void main()
{
   char sexe;
   unsigned int q;
   clrscr();
   printf ("Introdueixi el sexe (D/H): ");
   do
   { q = fscanf (stdin, "%c", &sexe);
     fflush(stdin);
   }
   while (q==0 || (sexe!='H' && sexe!='D'));
   printf ("Sexe introduït: %c\n",sexe);
}
```

Trobareu l'arxiu `u2n2p06.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Doncs aquí ens trobem amb una ingrata sorpresa depenent de la plataforma d'execució. El funcionament és el desitjat en certes plataformes (en Turbo C++, per exemple) i, en canvi, no és el desitjat en compiladors *gcc*.

El motiu radica en que la funció `fflush(canal)`, en principi, neteja el *buffer* associat al canal que se li indica, però en Linux només en fa cas pels canals de sortida i, la introducció pel teclat (`stdin`), és un canal d'entrada. Així doncs, ens trobarem que si executem el programa anterior `u2n2p06` i introduïm el mot `MARGARIDA`, el funcionament és:

- En plataforma DOS i/o Windows, el programa llegeix el primer caràcter `M` i efectua una neteja del *buffer*. En tractar-se d'un caràcter no esperat, no surt del `while` en espera que l'usuari introdueixi un caràcter correcte (`H` o `D`).
- En plataforma Linux, el programa llegeix el primer caràcter `M` i efectua una neteja del *buffer*... No es queixa de la instrucció `fflush(stdin)`, però no té cap efecte. Per tant, la resta de caràcters `ARGARIDA` continuen dins el *buffer* i els va gestionant 1 a 1 fins que arriba a la `D` que és un caràcter vàlid i mostra el missatge erroni final.

Què podem fer? Caldrà cercar una solució per a les plataformes Linux. Estaríem, llavors, desenvolupant programes diferents segons la plataforma, que és el que acostuma a succeir en la pràctica, però que no ens interessa per al desenvolupament dels exemples i exercicis d'aquest material.

Optem per una solució similar a la que varem adoptar per a poder utilitzar, en tots els exemples i exercicis d'aquest material, les funcions `clrscr()`, `gotoxy()`, `getch()` i `getche()` desenvolupades per Borland i que no existeixen en els compiladors *gcc* de Linux. La solució va consistir en dissenyar un arxiu `conio.h` que contingués una versió d'aquestes funcions per a la plataforma Linux.

De manera similar, doncs, utilitzarem en els nostres programes una funció de nom `neteja_stdin()` que netegi el *buffer* del canal `stdin` i que tindrem implementada de manera adequada a cada plataforma. Aquesta funció i d'altres que puguin sorgir en endavant, les inclourem en un arxiu de nom `compat.h` (per indicar compatibilitat).

Vegeu una versió de l'arxiu `compat-t.h` per a Turbo C++ i una versió de l'arxiu `compat-g.h` per a compiladors `gcc`, en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Ubicació de l'arxiu compat.h en els IDE Anjuta i Turbo C++

L'IDE Anjuta cerca els arxius declarats amb la directiva `#include` dins el directori `/usr/include` entre altres directoris.

L'IDE Turbo C++ cerca els arxius declarats amb la directiva `#include` dins el directori `xxx\include` on `xxx` correspon al directori on s'ha instal·lat Turbo C++.

Per garantir que tots els nostres programes trobin la versió de l'arxiu compat.h adequat a la plataforma, caldria ubicar la còpia dels arxius `compat-g.h` i `compat-t.h` amb nom `compat.h` en un dels directoris indicats.

Tenim, doncs, la nova codificació en C vàlida per ambdues plataformes:

```
/* u2n2p07.c */

#include <stdio.h>
#include <conio.h>
#include <compat.h>  /* per tenir compatibilitat entre plataformes */

void main(void)
{
   char sexe;
   unsigned int q;
   clrscr();
   printf ("Introdueixi el sexe (D/H): ");
   do
   { q = fscanf (stdin, "%c", &sexe);
     neteja_stdin(); /* definida dins compat.h */
   }
   while (q==0 || (sexe!='H' && sexe!='D'));
   printf ("Sexe introduït: %c\n",sexe);
}
```


Trobareu l'arxiu `u2n2p07.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

2.4. Estructura repetitiva `per`

En algunes situacions especials es coneix, a priori, la quantitat exacta de vegades que caldrà repetir el bucle. En tal cas s'agraeix l'existència de l'estructura repetitiva `per`.

L'estructura repetitiva `per` permet repetir un nombre determinat de vegades un conjunt d'instruccions.

Vegem com s'expressa aquesta estructura en pseudocodi, ordinograma i llenguatge C.

2.4.1. Estructura `per` en pseudocodi i ordinogrames

La sintaxi en pseudocodi és:

```
per comptador = valor_inicial fins valor_final [saltant salt] fer
   conjunt d'instruccions
fiper
```

No heu d'oblidar els punts següents:

- Cal tenir declarada una variable `comptador` (entera) de tipus comptador (el nom pot ser qualsevol dels permesos en el llenguatge) a la qual s'assigna automàticament el valor_inicial (enter) que consta a l'estructura.

- El valor de la variable `comptador` s'incrementa a cada final del bucle, és a dir, quan passa per la paraula reservada `fiper`, amb el contingut del `salt` (natural major de 0), el qual val la unitat en el seu defecte (acostuma a no ser obligatori).

- Abans d'iniciar tota execució del conjunt d'instruccions que forma el bucle (fins i tot la primera vegada) es comprova que el valor emmagatzemat a la variable `comptador` sigui menor o igual al valor_final que apareix a la instrucció. En cas que no ho sigui, s'acaba l'execució de l'estructura.

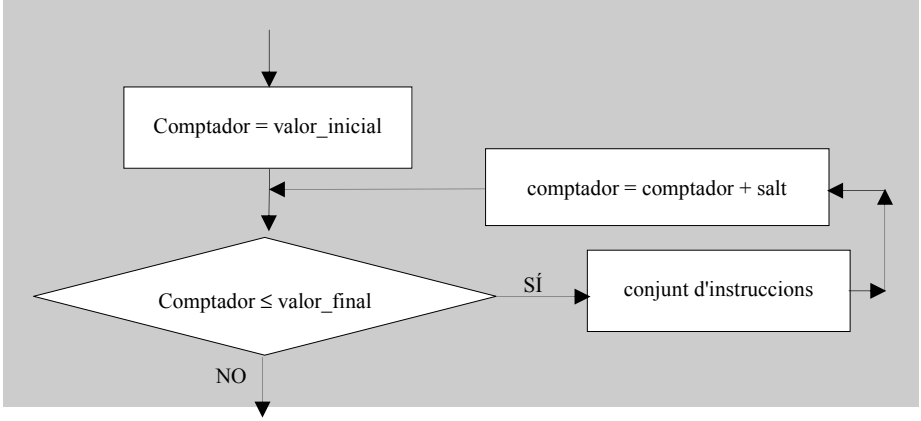
- No tots els llenguatges disposen de l'estructura `per`. En el seu defecte, cal utilitzar l'estructura repetitiva `mentre...fer` de la manera següent:

```
comptador = valor_inicial;
mentre comptador ≤ valor_final fer
   conjunt d'instruccions;
   comptador = comptador + salt;
fimentre
```

- Alguns llenguatges permeten la utilització d'un `salt` negatiu, de manera que la comprovació que es fa és, en tal cas, si el valor emmagatzemat a la variable comptador és major o igual al valor_final. En aquest crèdit considerarem únicament la possibilitat de salt positiu, amb la comparació referent a si el valor del comptador és menor o igual al valor_final.

La figura 8 mostra la representació de l'estructura repetitiva `per` en ordinograma.

Figura 8. Representació de l'estructura repetitiva `per` en un ordinograma.



2.4.2. Estructura `for` en llenguatge C

El llenguatge C disposa d'una estructura `per` molt més potent que la majoria de llenguatges. En realitat és com un `mentre...fer` camuflat. En efecte, la seva sintaxi és la següent:

```
for ([<Part 1>];[<condició_lògica>];[<Part 3>])
{  <conjunt d'instruccions a executar>;  }
```

Cal destacar-ne el següent:

- *<Part 1>* consisteix en un conjunt d'inicialitzacions de variables que han d'estar declarades. No tenen per què ser enteres.

- *<Part 3>* consisteix en instruccions que fan evolucionar el(s) valor(s) de les variables perquè la <condició lògica> deixi de ser certa i, per tant, s'acabi l'execució de l'estructura iterativa. És a dir, no és necessari que sigui un salt. Si ho és, cal escriure la instrucció d'increment corresponent.

- *<Part 1>* i *<Part 3>* poden estar formades per diferents instruccions; en tal cas, van separades per comes.

– En definitiva, *<Part 3>* podria obviar-se i constituir part del *<conjunt d'instruccions a executar>*, d'igual manera que a vegades el *<conjunt d'instruccions a executar>* podria obviar-se i situar-se com a components de *<Part 3>*. Però això no és lògic. Cal acostumar-se a situar les instruccions que provoquen el canvi de valor de la *<condició_lògica>* en la *<Part 3>* i el conjunt d'instruccions repetitiu en el *<conjunt d'instruccions a executar>*.

- És obligatori situar els dos "**;**" que separen les tres parts de la capçalera del `for` encara que alguna part no s'hi defineixi.

2.4.3. Exemples d'utilització de l'estructura `per`

Repetim el programa que consistia a comptar des d'1 fins a 100 situant 10 números a cada fila. La manera més simple de realitzar aquest programa és fent servir l'estructura `per`, ja que es coneix el nombre de vegades que cal repetir el procés (des d'1 fins a 100).

Algorisme en pseudocodi:

```
programa comptar_fins_a_100 és
   var n: natural;
   fivar
```

```
   netejar_pantalla;
   per n = 1 fins 100 fer
      escriure (n);
      si n mod 10 == 0 llavors saltar_línia; fisi
   fiper
fiprograma
```

Codificació en llenguatge C:

```
/* u2n2p08.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int n;
   clrscr();
   for (n=1; n<=100; n++)
   {  printf ("%5u",n);
      if (n % 10 == 0) printf ("\n");
   }
}
```

Trobareu l'arxiu `u2n2p08.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

Fixeu-vos en les variants d'escriptura següents, que permeten comprovar algunes de les observacions esmentades.

```
void main(void)
{
   unsigned int n=1;
   clrscr();
   for (; n<=100; n++)
   {  printf ("%5u",n);
      if (n % 10 == 0) printf ("\n");
   }
}
```

Si no calgués controlar que hi hagués 10 números per fila, podríem fins i tot escriure el codi de la manera següent:

```
void main(void)
{
   unsigned int n=1;
   clrscr();
   for (; n<=100; printf ("%5u",n), n++);
}
```

D'altra banda, és molt important recordar que en cas que s'utilitzi el `for` sense conjunt d'instruccions (ja que tot es resol en la seva capçalera), cal posar el "**;**" de després del parèntesi, perquè si no ho féssim, el programa traductor interpretaria que la instrucció següent formaria el cos del bucle.

2.5. Imbricació d'estructures repetitives

De la mateixa manera que succeïa en les estructures condicionals, en les repetitives s'acostuma a utilitzar la imbricació per a millorar la claredat i la llegibilitat dels programes.

Evidentment, també es poden imbricar estructures repetitives dins d'estructures condicionals i estructures condicionals dins d'estructures repetitives (aquest darrer cas ja l'hem estat fent en els exemples anteriors).

Quan imbriquem qualsevol tipus d'estructura iterativa o condicional en altres estructures només cal tenir en compte una condició: l'estructura interna ha d'estar inclosa completament dins l'externa.

Exemple

Volem fer un programa que permeti construir les taules de sumar amb una distribució quadriculada de 3 files i 3 columnes tal com es veu en la figura de la dreta.

```
1 + 0 =  1   2 + 0 =  2   3 + 0 =  3
1 + 1 =  2   2 + 1 =  3   3 + 1 =  4
1 + 2 =  3   2 + 2 =  4   3 + 2 =  5
1 + 3 =  4   2 + 3 =  5   3 + 3 =  6
1 + 4 =  5   2 + 4 =  6   3 + 4 =  7
1 + 5 =  6   2 + 5 =  7   3 + 5 =  8
1 + 6 =  7   2 + 6 =  8   3 + 6 =  9
1 + 7 =  8   2 + 7 =  9   3 + 7 = 10
1 + 8 =  9   2 + 8 = 10   3 + 8 = 11
1 + 9 = 10   2 + 9 = 11   3 + 9 = 12

4 + 0 =  4   5 + 0 =  5   6 + 0 =  6
4 + 1 =  5   5 + 1 =  6   6 + 1 =  7
4 + 2 =  6   5 + 2 =  7   6 + 2 =  8
4 + 3 =  7   5 + 3 =  8   6 + 3 =  9
4 + 4 =  8   5 + 4 =  9   6 + 4 = 10
4 + 5 =  9   5 + 5 = 10   6 + 5 = 11
4 + 6 = 10   5 + 6 = 11   6 + 6 = 12
4 + 7 = 11   5 + 7 = 12   6 + 7 = 13
4 + 8 = 12   5 + 8 = 13   6 + 8 = 14
4 + 9 = 13   5 + 9 = 14   6 + 9 = 15

7 + 0 =  7   8 + 0 =  8   9 + 0 =  9
7 + 1 =  8   8 + 1 =  9   9 + 1 = 10
7 + 2 =  9   8 + 2 = 10   9 + 2 = 11
7 + 3 = 10   8 + 3 = 11   9 + 3 = 12
7 + 4 = 11   8 + 4 = 12   9 + 4 = 13
7 + 5 = 15   8 + 5 = 13   9 + 5 = 14
7 + 6 = 13   8 + 6 = 14   9 + 6 = 15
7 + 7 = 14   8 + 7 = 15   9 + 7 = 16
7 + 8 = 15   8 + 8 = 16   9 + 8 = 17
7 + 9 = 16   8 + 9 = 17   9 + 9 = 18
```

Anàlisi funcional

Tenim un procés repetitiu (3 files) en què per a cada fila també tenim un altre procés repetitiu (3 columnes) i per cada columna un altre procés repetitiu, ja que tenim 10 files (processos imbricats).

Cal tenir en compte el procés que 'ha de seguir en cas de desitjar el resultat mitjançant una impressora.

Quan cal presentar una informació en pantalla en pseudocodi, es pot utilitzar la instrucció `cursor (col, fil)` per situar el cursor a la posició desitjada i, posteriorment, escriure-hi la informació que correspongui. En canvi, quan cal presentar una informació per impressora, s'ha d'enviar les instruccions d'escriptura en el mateix ordre en què s'han de llegir en el paper (d'esquerra a dreta i de dalt cap a baix). En pantalla, es pot seguir qualsevol ordre; en impressora, no.

Anàlisi orgànica

Considerarem la presentació per pantalla, però sense utilitzar la instrucció `cursor`, és a dir, de la mateixa manera que hauríem de seguir per presentar-la per impressora. No sabem, encara, com s'envia la informació per impressora i, per tant, l'escriurem per pantalla.

Reduirem a dos els tres processos iteratius imbricats descrits a l'anàlisi funcional. Primer considerarem el procés repetitiu consistent a pintar les tres files de la quadrícula i, per cada fila, considerarem el procés repetitiu consistent a pintar les deu files de la taula, escrivint en paral·lel les tres columnes.

L'algorisme en pseudocodi és:

```
programa taules_de_sumar és
var      f: natural; /* per controlar les tres files de la quadrícula */
         t: natural; /* per controlar les taules de cada fila */
         n: natural; /* per controlar les 10 files de cada taula */
   fivar
   netejar_pantalla;
   escriure_línia_superior_de_la_quadrícula;
   per f = 1 fins 3 fer
      si f == 2 o f == 3 llavors escriure_línia_separació_quadrícula; fisi
      t = 3 * f - 2;
      /* a la fila 1 (f == 1), corresponen les taules de l'1, 2 i 3;
         a la fila 2 (f==2), corresponen les taules del 4, 5 i 6;
         és a dir, a la fila f, corresponen les taules de 3*f-2, 3*f-1 i 3*f */
      per n = 0 fins 9 fer
         escriure ("|", t, "+", n, "=", t+n, "|", t+1, "+",n, "=", t+1+n, "|", t+2, "+", n, "=",
                   t+2+n, " |");
         /* escrivim les tres taules d'una mateixa fila en paral·lel;
            caldrà anar amb compte que tot quedi ben encolumnat */
      fiper
   fiper
   escriure_línia_inferior_de_la_quadrícula;
fiprograma
```

Quan es treballa en pseudocodi, no cal perdre temps en floritures respecte de la presentació. El que ha de quedar clar és la informació que es presenta i l'ordre en què es presenta. Si cal tenir present algun fet per al moment de codificació s'hi escriu com a comentari, tal com s'ha fet.

Quan codifiquem en llenguatge C hem de fer servir els formats d'especificació en la funció `printf` per garantir que totes les dades queden convenientment encolumnades.

```
/* u2n2p09.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int f,t,n;
   clrscr();
   printf ("┌──────────────────────────────────────────┐");
   printf ("\n");
   for (f=1;f<=3;f=f+1)
   {
      if (f==2 || f==3)
      printf ("├──────────────┼──────────────┼──────────────┤");
      printf ("\n");
      t=f*3-2;
      for (n=0;n<=9;n++)
      printf ("│ %2d + %2d = %2d │ %2d + %2d = %2d │ %2d + %2d = %2d"\
              " │\n", t, n, t+n, t+1, n, t+1+n, t+2, n, t+2+n);
   }
   printf ("└──────────────────────────────────────────┘");
   printf ("\n");
}
```

Trobareu l'arxiu `u2n2p09.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

La impressió de les tres taules en paral·lel s'hauria pogut programar utilitzant un altre `per` (`for`) imbricat. Vegem-ho directament en llenguatge C:

```
/* u2n2p10.c */

#include <stdio.h>
#include <conio.h>

void main(void)
{
   unsigned int f,t,n,c;
   clrscr();
   printf ("┌──────────────────────────────────────────┐");
   printf ("\n");
   for (f=1;f<=3;f=f+1)
   {
      if (f==2 || f==3)
      printf ("├──────────────┼──────────────┼──────────────┤");
      printf ("\n");
      t=f*3-2;
      for (n=0;n<=9;n++)
      {
         for (c=0; c<3; c++)
            printf ("│ %2d + %2d = %2d ", t+c, n, t+c+n);
         printf ("│\n");
      }
   }
   printf ("└──────────────────────────────────────────┘");
   printf ("\n");
}
```

Trobareu l'arxiu `u2n2p10.c` en el contingut "Codi font en C dels programes desenvolupats en material paper" de la web d'aquest crèdit.

2.6. Trencament d'estructures repetitives

Sabem que les estructures repetitives consisteixen en la repetició de l'execució d'un conjunt d'instruccions sota la supervisió d'una condició lògica que s'avalua abans o després de l'execució del conjunt d'instruccions i determina la continuació o l'avortament del procés repetitiu. Aquest és el funcionament desitjable en programació, ja que assegura que la condició lògica és l'únic que governa el procés.

Alguns llenguatges, però, incorporen en les seves sentències repetitives la possibilitat de trencar el procés repetitiu sense necessitat d'esperar la condició lògica. No és una bona pràctica acostumar-se a fer servir aquestes possibilitats, ja que treu llegibilitat i claredat al programa.

Quan un programa no funciona com se suposa que hauria de funcionar i cal fer el seguiment del flux d'execució, en arribar a un procés repetitiu, el programador agrairà sempre que aquest sigui governat únicament i exclusivament per la condició lògica. Si dins del propi procés repetitiu s'ha utilitzat sentències de trencament que possibiliten l'escapatòria del bucle sense haver-lo acabat i sense la supervisió de la condició lògica, serà més difícil fer-ne el seguiment corresponent.

Davant el raonament anterior us deu sorgir la pregunta: i doncs, per què els llenguatges incorporen aquestes possibilitats?

Unes línies més amunt hem assegurat que no és una bona pràctica acostumar-s'hi, però no ho hem prohibit. Aquestes possibilitats tenen llicència d'utilització en el control de les excepcions. Suposem un procés repetitiu consistent a processar un seguit d'informació que li arriba per un determinat canal i que té una marca de final. Suposem que el programa ha de controlar la possibilitat, remota però real, de rebre una determinada marca que ha de provocar l'avortament immediat del procés repetitiu amb algun tractament especial per a tal ocasió.

El bon programador (cent per cent estructurat) programaria el següent:

```
   rebre_la_primera_informació;
   mentre no_sigui_la_marca_de_final i no_sigui_la_marca_remota fer
      tractament_normal;
      rebre_la_següent_informació;
   fimentre
/* si hem sortit del mentre és perquè s'ha deixat de
verificar la condició lògica, la qual cosa només pot passar, com que
es tracta d'una conjunció, perquè s'ha deixat de verificar algun dels
seus components, és a dir, o s'ha arribat a la marca de final o s'ha
arribat a la marca de la possibilitat remota; quan sortim del mentre
esbrinem el motiu amb una estructura condicional */
if és_la_marca_remota llavors tractament_especial; fisi
```

És a dir, és la condició lògica la que governa la sortida del bucle. Ara bé, si la immensa majoria de les vegades no es produeix la possibilitat remota, un programador pot considerar més clar no incloure aquesta possibilitat a la condició lògica amb el raonament següent: la condició lògica farà referència al flux normal d'execució i les possibilitats remotes les controlaré dins del flux amb sortides per la porta falsa. En tal cas, el codi anterior quedaria:

```
rebre_la_primera_informació;
mentre no_sigui_la_marca_de_final fer
   si és_la_marca_remota llavors
      tractament especial;
      trencar;
   fisi
   tractament_normal;
   rebre_la_següent_informació;
fimentre
```

En pseudocodi incorporarem les dues possibilitats més normals de trencament d'estructures repetitives: `trencar` i `continuar`.

La sintaxis de cadascuna és, simplement:

```
trencar;
continuar;
```

Ja hem vist la utilització de `trencar` en el darrer exemple. Provoca la sortida de l'estructura repetitiva més interna on es troba i l'execució continua a la primera instrucció posterior a l'estructura repetitiva.

La sentència `continuar` també provoca el trencament del flux d'execució del bucle, que continua a l'avaluació de la condició lògica perquè sigui aquesta la que decideixi la continuïtat o no del procés repetitiu (en cas afirmatiu es torna a començar el bucle amb els valors actuals de les variables, és a dir, no es continua en la instrucció següent on s'havia produït el trencament). És a dir, aquesta sentència obliga a executar la següent iteració del bucle.

En cas de tenir imbricació d'estructures repetitives, aquestes sentències només fan referència a l'estructura repetitiva més interna.

En ordinogrames no hi ha una forma especial per a representar aquestes instruccions. En el flux d'execució del bucle imposarem la continuació del flux per on interessi amb la utilització d'una fletxa. En el cas de `trencar` aniria a la instrucció posterior al bucle i en el cas de `continuar` aniria a l'avaluació de la condició lògica.

El llenguatge C disposa de les dues sentències vistes en pseudocodi. Són:

```
break;          /* Equivalent a trencar */
continue;       /* Equivalent a continuar */
```

2.7. El salt incondicional

Explicarem aquesta possibilitat perquè la majoria de llenguatges la incorporen, però cal dir d'entrada que queda totalment prohibida la seva utilització en aquest crèdit, ja que és l'antítesi de la programació estructurada. (!!)

Aquesta instrucció permet trencar l'execució seqüencial del flux d'execució, i provocar que aquest continuï en una altra instrucció de programa a la qual es fa referència mitjançant una etiqueta.

En pseudocodi considerarem la sintaxi següent:

```
anar_a <etiqueta>;
```

en què etiqueta ens envia a una línia que conté la declaració de l'etiqueta:

```
etiqueta <etiqueta>
```

El flux d'execució continuarà per la línia següent a la línia on es troba la declaració de la corresponent <etiqueta>.

La utilització de la instrucció `anar_a` presenta els inconvenients següents:

- Dificulta el seguiment del programa en què es fa servir aquesta instrucció
- És innecessari, ja que es pot substituir per la correcta utilització de les estructures repetitives i condicionals.
- Crea problemes en l'execució del programa quan es fan salts a l'interior de les estructures repetitives i condicionals.

Hi ha, però, una escletxa per justificar la seva utilització: si cal abandonar l'execució d'una estructura repetitiva amb grau elevat d'imbricació quan el flux es troba dins d'estructures repetitives internes, la sentència `trencar` no aporta la funcionalitat necessària, ja que es limita únicament a un nivell d'imbricació. En tal cas, queda justificada la utilització de la sentència `anar_a`.

El llenguatge C disposa d'aquesta sentència. La seva sintaxi és la següent:

```
goto <etiqueta>;
```

la qual cosa obliga que hi hagi alguna instrucció del programa etiquetada convenientment. La sintaxi d'utilització d'una etiqueta és:

```
<etiqueta>: [instrucció]
```

La línia que conté l'etiqueta pot anar o no acompanyada d'una instrucció. Si hi va, el flux d'execució continuarà per aquesta instrucció; en cas contrari, es continua per la primera instrucció que hi hagi a les línies següents.