Unitat Formativa 1. Programació estructurada

Abans de començar a descriure els elements de programació amb els que podem construir programes, farem unes introducció als tipus de llenguatges de programació, els tipus de programes que construïm amb ells, així com els entorns de desenvolupament utilitzats.

Classificació dels llenguatges de programació en generacions

1ª Generació	llenguatge maquina, és un llenguatge binari de baix nivell que es basa a la codificació amb ceros i uns, que és lo que realment "entén" directament la computadora.
2ª Generació	llenguatge assemblador, un altre llenguatge de baix novell però on apareixen les primeres instruccions formades per paraules (paraules clau), on cada instrucció equival a una seqüència de bits generada en codi maquina
3ª Generació	Aparició dels llenguatges d'alt nivell, on una instrucció pot equivaldre a un conjunt d'instruccions per al processador). Faciliten la escriptura i lectura de els programes per que el llenguatges és més similar a l'humà. Aquests llenguatges poden ser no estructurats: **ALGOL, FORTRAN, COBOL, BASIC**... O estructurats:**, PASCAL, ADA, C MODULA-2**... A més dels orientats a objectes **C++, EIFFEL, SMALLTALK, OBJECT PASCAL** ... I els especialitzats en determinades tasques: **HTML**, **JAVA, JAVA SCRIPT, PHP**...
4ª Generació	Llenguatges orientats a tasques més específiques com a la gestió de bases de dades (llenguatges SQL) i llenguatges de prototipus, que a partir d'un prototipus comencen a generar el codi del programa. Com per exemple els llenguatges de programació gràfica com els **Visual Basic**, **Visual C**, **Delphi**, etc.

Classifiquem els llenguatges orientats a Intel·ligència Artificial com de 5ª Generació.

Tipus de processament o execució d'un programa

Processament per lots	De tipus seqüencial, s'executa sempre en temps real una instrucció rere l'altre fins que finalitza el programa. És el mètode més comú, un exemple seria un programa per realitzar backups.
Multiprogramació	Execució simultània de varis programes que es transfereixen informació, compartint els recursos del PC (parlem de "temps compartit"). Un exemple serien el S.O. i els seus diferents processos.
Processament interactiu	Programa i usuari estan en comunicació, per que l'usuari ha de proporcionar les dades o paràmetres necessaris per a la execució. Un exemple seria el programa d'un caixer electrònic.

Entorns integrats de desenvolupament (IDE). Projectes

Per realitzar una aplicació disposem de programes específics, que poden ser tan bàsics com un simple compilador que ens crearà el executable a partir d'un codi font, escrit amb un editor independent del compilador, o més complerts com els IDE's (Integrated Development Enviroment) que incorporen els elements necessaris per obtenir el programa:

- Un entorn de desenvolupament gràfic, **GUI** (Graphic User Interface), que ens facilita l'accés a totes les eines i components del IDE mitjançant els clàssics menús, botons i finestres.
- Un editor de text, amb la possibilitat de que sigui el suficientment avançat com per ajudar a la escriptura de codi.
- Poden oferir un ampli ventall de solucions, i amb diferents llenguatges de programació.
- Un compilador, que segons el IDE pot treballar amb diferents llenguatges de programació (C, C++, C#, Java, Visual Basic...).
- Un entorn de depuració, que com a mínim constarà de les següents capacitats:
  - Execució pas a pas (una per una) de les sentències del programa.
  - Visualització del valor de les variables durant l'execució del programa (**watching**).
  - Parada i verificació de variables en punts concrets del programa (**breakpoints**).

Alguns dels IDE's propietaris més coneguts com **Visual.NET** i **Visual Studio Code** de **Microsoft,** ofereixen una amplia gama de recursos i una gran col·lecció d'assistents que poden ajudar a desenvolupar aplicacions senzilles en molt poc temps (el problema dels assistents és que generen molt codi innecessari i això repercuteix negativament a la eficàcia del programa).

Alguns exemples serien dels IDE's de codi obert més coneguts serien:

- **NetBeans** (www.netbeans.org/): un entorn que suporta una gran varietat de llenguatges, Java, PHP, Perl, Ruby, Python... i instal·lant un complement específic també suporta C/C++ amb tots els avantatges que proporciona aquest conegut IDE.

- **Eclipse** (http://www.eclipse.org/): l'altre gran entorn de desenvolupament de la comunitat lliure, igual que NetBeans suporta una gran quantitat de llenguatges de programació (pràcticament els mateixos) i també incorpora un entorn gràfic (GUI) de molta qualitat.

¿Què es un projecte?

Actualment els programes es desenvolupen a partir de la integració d'un conjunt de fitxers de diferents tipus i continguts anomenat "projecte". El fitxer que contindrà el nostre codi font, el que hem d'escriure per realitzar una determinada tasca, serà un fitxer més del projecte, però el més important per que serà el que tindrà la solució específica que necessitem, però no per això podem prescindir de la resta.

Característiques del llenguatge PHP

PHP és un dels llenguatges més populars al desenvolupament d'aplicacions web. Podem trobar abundant documentació a la web, però aquests son alguns dels enllaços més recomanables:

Manual online traduït al castellà: https://www.php.net/manual/es/

Tutorials pràctics: https://www.w3schools.com/php/
http://www.mclibre.org/consultar/php/
https://www.guru99.com/php-tutorials.html
https://diego.com.es/formularios-en-php

Podem iniciar-nos al llenguatge algun dels editors de codi PHP online, abans de crear el nostre entorn de desenvolupament:

https://onecompiler.com/php/
https://paiza.io/es/projects/new
https://www.tutorialspoint.com/execute_php_online.php

Característiques principals

- Es tracta d'un tipus de llenguatge **case-sensitive**, que vol dir que distingeix entre majúscules i minúscules alhora d'assignar noms a variables i funcions.

- Com veurem al següent apartat les variables han de començar pel caràcter **$**

- Si el nom de la variable es compost per dos conceptes o més es recomana seguir la pràctica de fer servir la majúscula com a "element de distinció". Per exemple es recomana fer servir un nom com *$edatFill* enlloc de *$edat_fill*.

- Com a forma de treball els noms que utilitzen el caràcter _ o s'escriuen totalment en majúscules són les **variables predefinides** per a usos específics de PHP, i que estudiarem més endavant. Alguns exemples: *$_SESSION, $_GET, $_POS* …

- Utilitza un **tipat dinàmic**, anomenat també **tipat dèbil**.

  Cal explicar que un llenguatge amb **tipat estàtic** o **tipat fort** es aquell que quan declara que una variable es d'un determinat tipus de dada (un tipus numèric o de text) queda fitxat per a tot el programa, i ja no es pot canviar. Alguns exemples molt coneguts d'aquests llenguatges són C/C++, C#, Java...

  Un llenguatge de tipat dinàmic com PHP no necessita que li indiquem el tipus de dada pel qual definim la variable, en temps d'execució s'adapta al contingut de la variable, i si es necessari li canvia el tipus. Dinàmicament una variable numèrica es pot convertir en una variable per guardar text.

En llenguatges que com propòsit important sigui presentar informació i no aturar el flux de comunicació per culpa d'un error, aquesta forma de treballar amb el tipat té sentit. Hem de ser conscients que es tracta de presentar resultats que es barrejaran amb el codi html de la pàgina web. No volem aturar la càrrega de la pàgina.

Aquesta filosofia fa que PHP es preocupi constantment de adaptar operacions de la millor forma possible encara que per fer-lo hagi de realitzar transformacions que ens poden confondre. Així automàticament sumarà cadenes de text si el seu contingut es numèric com si el seu tipus fos numèric, però també sumarà un valor boolean a un valor numèric, ...

I diversos casos similars on podem trobar aquesta característica com molt pràctica o com un desastre sinó tenim cura de que estem realitzant en tot moment i quines operacions tenen sentit o quines no, perquè totes retornaran un resultat però hem de mirar que sigui correcte.

- Per escriure codi PHP sempre hem de començar amb la etiqueta **<?php**

  Si es tracta d'un fitxer amb extensió .php no cal tancar la etiqueta amb un ?**>**

  De fet, ens indicarà que no correcte. Però si el codi php està incrustat dins del codi html d'una pàgina web si caldrà tancar la etiqueta.

  Exemple:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset='UTF-8'>
        <title>DEMO PHP</title>
    </head>
    <body>
        <?php
            $x = 3;
            $y = "2";
            $z = $x + $y;
            echo $z;
            echo "<br />";
            $z = $x . $y;
            echo $z;
            echo '<br />';
        ?>
    </body>
</html>
```

Estructura d'un programa informàtic

Definició de variables. Tipus i utilitat.

Les variables són objectes de programa que fem servir per guardar valors temporalment. Es tracta d'un tipus de reserva de memòria (memòria RAM) per guardar les dades del programa, i com diu el seu nom el seu contingut pot variar durant l'execució del programa,

Una variable serà utilitzada tant per guardar dades que procedeixen de fora del programa com pels resultats de les operacions internes del mateix, i el seu valor pot ser modificat en qualsevol moment durant la seva execució. Una vegada finalitzat el programa la reserva de memòria RAM "caduca" i tots els valors es tornen inaccessibles, els podem donar per perduts.

Identificadors: nom assignat a les variables i constants, amb PHP sempre ha de començar pel símbol **$** i el primer caràcter ha de ser una lletra, encara que després es poden afegir dígits: ***num1*** es correcte, però ***1num*** no. El llenguatge PHP es **case-sensitive**: distingeix entre majúscules i minúscules (num1 i Num1 es consideren 2 variables diferents, així com qualsevol combinació de majúscules i minúscules de "num").

Les variables poden ser dels següents tipus:

- **Boolean:** només es pot guardar un 0 ó un 1 (fals o vertader). *$cert = true; //o false*
- **Numèric:** només permet guardar números (amb o sense signe), cal especificar si són de tipus **sencers** (sense decimals) o amb **decimals**

Treballant amb PHP podem escollir entre:

integer	contindrà un sencer	*$valor = 377;*
float	contindrà un número amb decimals	*$valor = 3.77;*

Un cas especial es precisament una variable que no varia de valor, sinó que és constant i per aquest motiu les anomenem **constants.** Les utilitzem per marcar límits o establir valors fixes:

```
const MAX_VALUE = 1000;          //el seu nom en majúscules per fer-lo destacar
define (MAX_VALUE, '1000');      //una alternativa amb la funció define
```

Mida i rang de les dades de tipus numèric (programes 32 bits)

TIPUS	Mida	Rang	Aplicacions pràctiques
int	32	-2.147.483.648 a 2.147.483.647	Comptadors, petits números, control de bucles
float	32	$3.4 \cdot 10^{-38}$ a $3.4 \cdot 10^{38}$	Científic (precisió fins a 7 dígits)

- **Vectors numèrics (Arrays) :** Són un conjunt finit de dades d'un mateix tipus (sencer o decimal). Tots els elements del vector es referencien a partir del nom del vector, així que per treballar amb un element en particular cal especificar a més del nom del vector la seva posició dins del mateix, per això farem servir un subíndex dintre d'uns claudàtors (**[ ]**). Aquest subíndex sempre ha de ser un valor de tipus sencer.

Consideracions específiques dels vectors:

- La primera posició o el primer valor guardat a un vector es troba a la posició 0, pot sorprendre que no es comenci pel 1, però la raó es la següent, el numero sencer que va a dins dels claudàtors indica desplaçament respecte a la primera posició, per tant quan entre claudàtors posem un 0 en realitat volem indicar que no cal fer cap desplaçament, que volem treballar amb el primer element.
- Així mateix si posem el numero 5 a dins dels claudàtors el que volem indicar es que cal desplaçar-se 5 posicions respecte a la inicial per ubicar-nos a sobre del valor que volem, i en aquest cas hi serem a sobre de la sisena posició.
- Es responsabilitat del programador ni intentar accedir a posicions del vector per sobre de la seva mida (si el vector té 10 elements les posicions que podem adreçar van de la 0 a la 9, mai intentarem accedir a les posicions 10 cap a munt), per que es pot produir un error greu que aturarà el programa.
- Excepte al moment de la creació o definició del vector, sempre haurem de carregar els valors del vector o presentar-los, d'un en un (posició per posició).

Exemple: *$valors = array(1,2,3,4,5,6,7);* O bé senzillament: *$valors = [1,2,3,4,5,6,7];*

Per accedir als valors hem d'indicar la posició amb el índex: *$valors[0];*

Per afegir valors: *$valors[] = 9;*

Per conèixer quants elements inclou l'array utilitzem una funció: *$numElems = count($valors);*

Vectors Multidimensionals: Són vectors de més d'una dimensió, per referir-nos a un element o posició en concret hem de fer servir més d'un subíndex. Amb tractament d'imatges s'utilitza molt aquest tipus de vectors ja que amb un vector de 2 subíndexs (imaginem una matriu) podem treballar amb imatges 2D, amb 3 subíndexs (imaginem un cub de Rubik) amb imatges 3D, amb 4 subíndexs podem incloure com afecta pas del temps a una representació en 3D...

```
$valors = [  [1, 2, 3, 4, 5, 6, 7],
             [8, 9, 10, 11, 12, 13, 14],
             [15, 16, 17, 18, 19, 20, 21],
             [22, 23, 24, 25, 26, 27, 28]   ];
```

Per accedir als valors ara necessitem 2 subíndex: *$valors[2][4];*

- **Cadenes de caràcters (strings)**: es tracta d'un conjunt de caràcters, amb els podem treballar individualment (caràcter a caràcter, o lletra a lletra) o directament amb el conjunt complert.

  Els caràcters de la cadena es poden referenciar individualment a partir del nom de la variable i un subíndex dintre dels claudàtors **[ ]**, com els vectors numèrics. El subíndex sempre ha de ser un valor de tipus sencer, i igual que amb els vectors numèrics, la primera posició es la 0.

  $linea = "aixo es un exemple";

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	i	x	o		e	s		u	n		e	x	e	m	p	l	e	\0

  La particularitat que presenten les cadenes respecte als vectors numèrics es que per als humans llegir o introduir de cop un conjunt de les posicions de la cadena té sentit, ja que equival a treballar amb paraules, o frases, o línies de comandaments. Així podríem imprimir per pantalla el contingut de la cadena i entendre perfectament el significat, cosa que seria poc pràctic amb un vector numèric.

  echo $linea;
  echo $linea[6];

  Un vector de caràcters de 2 dimensions equival a definir un conjunt de cadenes de caràcters, i pot guardar la informació que podem trobar a una pàgina de text o el contingut d'un fitxers de dades.

```
$pagina= [
            ["aixo es un exemple"],
            ["de com assignar valors"]
];
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	a	i	x	o		e	s		u	n		e	x	e	m	p	l	e	\0				
1	d	e		c	o	m		a	s	s	i	g	n	a	r		v	a	l	o	r	s	\0

  Un vector de caràcters de 3 dimensions serviria per guardar el contingut de un llibre.

  PHP permet treballar amb **strings** indicant el seu valor entre **cometes simples (' ') o dobles (" ")**. La diferencia únicament es troba en que amb cometes dobles ve implícit el mecanisme de substitució de variables pel seu valor, mentre que amb cometes simples aquest mecanisme no està activat per defecte.

```
echo "el valor de la variable es $num";      //mostrarà el seu valor
echo 'el valor de la variable es $num';       //mostrarà $num, no el seu valor
```

TAULA DE CODI ASCII (equivalència valor sencer-caràcter)

0=	1=☺	2=☻	3=♥	4=♦	5=♣	6=♠	7=	8=	9=
10=	11=	12=	13=	14=♫	15=☼	16=►	17=◄	18=↕	19=‼
20=¶	21=§	22=▬	23=↨	24=↑	25=↓	26=→	27=←	28=∟	29=↔
30=▲	31=▼	32=	33=!	34=“	35=#	36=$	37=%	38=&	39='
40=(	41=)	42=*	43=+	44=,	45=-	46=.	47=/	48=0	49=1
50=2	51=3	52=4	53=5	54=6	55=7	56=8	57=9	58=:	59=;
60=<	61==	62=>	63=?	64=@	65=A	66=B	67=C	68=D	69=E
70=F	71=G	72=H	73=I	74=J	75=K	76=L	77=M	78=N	79=O
80=P	81=Q	82=R	83=S	84=T	85=U	86=V	87=W	88=X	89=Y
90=Z	91=[	92=\	93=]	94=^	95=_	96=`	97=a	98=b	99=c
100=d	101=e	102=f	103=g	104=h	105=i	106=j	107=k	108=l	109=m
110=n	111=o	112=p	113=q	114=r	115=s	116=t	117=u	118=v	119=w
120=x	121=y	122=z	123={	124=|	125=}	126=~	127=⌂	128=Ç	129=ü
130=é	131=â	132=ä	133=à	134=å	135=ç	136=ê	137=ë	138=è	139=ï
140=î	141=ì	142=Ä	143=Å	144=É	145=æ	146=Æ	147=ô	148=ö	149=ò
150=û	151=ù	152=ÿ	153=Ö	154=Ü	155=ø	156=£	157=Ø	158=×	159=ƒ
160=á	161=í	162=ó	163=ú	164=ñ	165=Ñ	166=ª	167=º	168=¿	169=®
170=¬	171=½	172=¼	173=¡	174=«	175=»	176=░	177=▒	178=▓	179=|
180=┤	181=Á	182=Â	183=À	184=©	185=╣	186=║	187=╗	188=╝	189=¢
190=¥	191=┐	192=└	193=┴	194=┬	195=├	196=─	197=┼	198=ã	199=Ã
200=╚	201=╔	202=╩	203=╦	204=╠	205=═	206=╬	207=¤	208=ð	209=Ð
210=Ê	211=Ë	212=È	213=ı	214=Í	215=Î	216=Ï	217=┘	218=┌	219=█
220=▄	221=¦	222=Ì	223=▀	224=Ó	225=ß	226=Ô	227=Ò	228=õ	229=Õ
230=µ	231=þ	232=Þ	233=Ú	234=Û	235=Ù	236=ý	237=Ý	238=¯	239=´
240=-	241=±	242=‗	243=¾	244=¶	245=§	246=÷	247=¸	248=°	249=¨
250=·	251=¹	252=³	253=²	254=■					

Conversions de tipus de dades

La conversió d'un tipus de dada a un altre (conegut també com **casting**) es realitza durant l'execució del programa per a augmentar la precisió i coherència d'algunes operacions. Podem trobar-nos amb 2 tipus de conversions:

- **Implícita: PHP** *asigna* automàticament els tipus de les variables en funció del valors rebuts o la operació a realitzar amb ells. Com es va veure a les primeres pràctiques.

- **Explícita:** conversió "forçada", redefinim el tipus de dada per donar coherència a l'expressió, però cal tenir en comte que si convertim una dada de tipus *float* a *int* es perden els decimals i inclús en funció del valor del float s'eliminaran els bits de major pes i es perdrà la part més important de la informació. Deixant de banda els casos lògics de cadenes de text a valor numèrics que realment no contenen. Exemples de conversió explícita:

```
$mitja = (int) (100/3);
$numInt = (int) 8666000000;                    //no té sentit, perdem informació
$numFloat = (float) (2345);
$numInt = (int) ("Prueba este casting");       //no té sentit, es incompatible
```

Programació estructurada

Realitzar un programa requereix d'una lògica aplicada a la adequada seqüència de les operacions a realitzar i els probables resultats que podem obtenir durant l'execució del programa per preveure les accions a portar a terme a cada pas. Per tant requereix d'un esforç per esbrinar l'ordre de les accions, unes estructures lògiques que recolzin al programador i una certa previsió dels possibles cassos davant els quals el programa ha de prendre decisions.

Regles de la Programació Estructurada:

- Fomentar la simplicitat del codi, així el desenvolupament de qualsevol programa només farà servir 3 tipus d'estructures per controlar el seu flux d'instruccions:
  - ***De seqüència***: Executa les instruccions del programa una rere l'altre tal com i són escrites.
  - ***De selecció o Control***: Permet realitzar "bifurcacions" a la execució del codi en funció de l'acompliment o no d'una condició, d'aquesta manera dotem al programa d'una certa flexibilitat per que variem les instruccions a executar en funció del resultats obtinguts.
  - ***De repetició***: Permet repetir un bloc d'instruccions una vegada rere l'altre fins que es compleix una determinada condició. Aquesta estructura permet reduir el nombre de línies del programa.

- Les estructures de selecció o control i les de repetició només es poden "combinar" de 2 formes:
  - ***Apilant***: col·locant una a continuació de l'altre, quan finalitza o sortim d'una començarem o entrarem a l'altre.
  - ***Nidant***: col·locant una a dins de l'altre, amb la possibilitat nidar més d'una estructura apilant-les o bé tenir diversos nivells de nidació.

Introducció a l'algorísmica

Un algorisme és la representació del procés a seguir per a resoldre un determinat problema, establint perfectament les accions a executar i l'ordre d'execució. Es pot representar gràficament per que sigui més fàcilment assimilable el seu funcionament mitjançant els **Diagrames de flux de dades**, que permeten veure el circuit que segueixen les dades des de la seva declaració, establint la seqüència de les operacions.

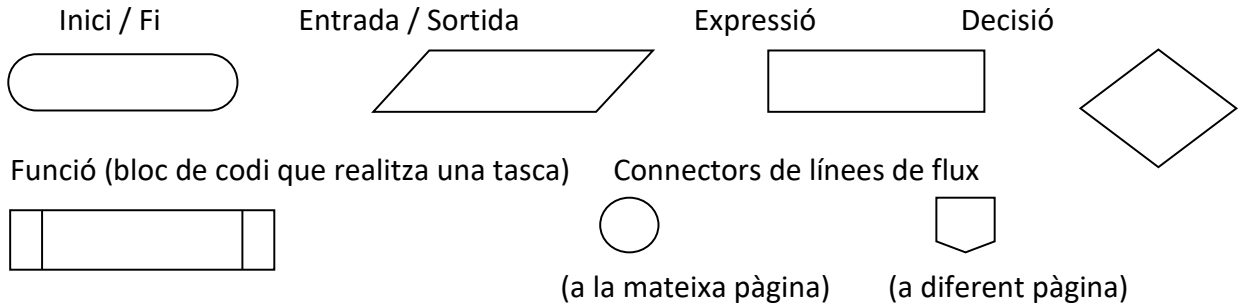
Un algorisme no dona la solució definitiva, i per això es menysprea quan només es busquen solucions ràpides que no ens facin pensar, però la programació requereix d'un mínim d'anàlisi i reflexió, i les eines gràfiques han estat des de sempre les més pràctiques per aquestes tasques.

Ordinogrames: s'utilitzen a la primera etapa de programació per representar gràficament pas a pas la seqüència lògica de les instruccions del programa. Tot ordinograma ha de tenir només un inici i només un fi de programa i al mig la seqüència d'instruccions.

Recomanacions per realitzar un ordinograma:

a) Fe servir línies rectes
b) El símbol d'inici del programa sempre a la part superior i el de fi a la inferior, i només un de cada
c) Que el flux d'operacions sigui de dalt a baix i d'esquerre a dreta.
d) El ordinograma ha de ser lo més simètric possible.
e) Les expressions utilitzades no cal que siguin de cap llenguatge de programació.
f) Intentar no creuar línies de flux.
g) Les instruccions tindran el mínim de detall

El ordinogrames fan servir generalment els següents símbols:



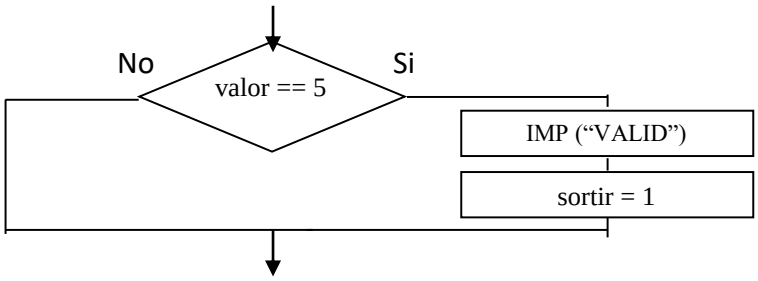
Estructures de selecció

Avaluen una expressió lògica o condició per seleccionar quin dels blocs específics d'instruccions, cal executar en funció del resultat obtingut al avaluar aquesta expressió.

Diferenciarem entre:

Estructures de Selecció simple: Si es compleix la condició o grup de condicions establertes s'executa un determinat bloc d'instruccions, si no es compleix no s'executa cap bloc d'instruccions dins d'aquesta estructura i continua el programa.

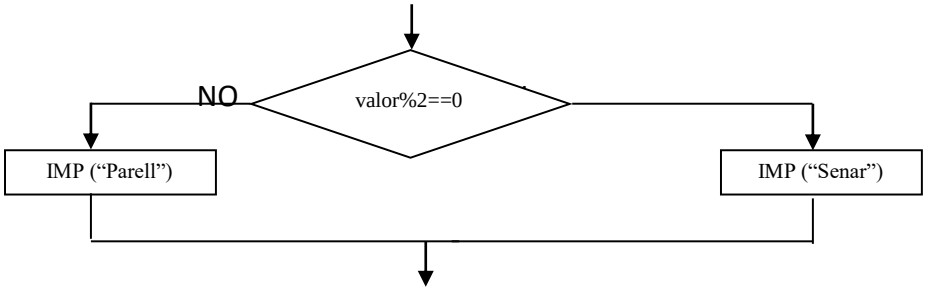
Representació amb un ordinograma:




```
if ($valor == 5) {
  echo "valid";
  $sortir = 1;
}
```

Estructures de Selecció doble: Si es compleix la condició o grup de condicions establertes s'executa un determinat bloc d'instruccions, si no es compleix s'executa l'altre bloc d'instruccions.

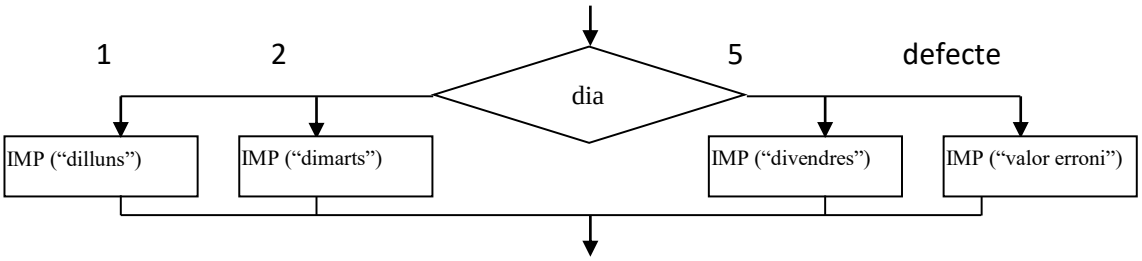
Representació amb un ordinograma:




```
if ($valor % 2 == 0) {
   echo "Parell";
}
else {
   echo "Senar";
}
```

Estructures de Selecció Múltiple: en funció del valor d'una variable del programa s'executarà un bloc d'instruccions específiques, si no coincideix amb cap dels valors que hem previst i no tenim res preparat per aquest cas es passarà per aquesta estructura sense realitzar cap operació.

Representació amb un ordinograma:




```
switch ($valor) {
   case 1: echo "dia $valor: dilluns";
           break;
   case 2: echo "dia $valor: dimarts";
           break;
   case 3: echo "dia $valor: dimecres";
           break;
   case 4: echo "dia $valor: dijous";
           break;
   case 5: echo "dia $valor: divendres";
           break;
   case 6: echo "dia $valor: dissabte";
           break;
   case 7: echo "dia $valor: diumenge";
           break;
  default: echo "Error al indicar un codi per al dia";
}
```

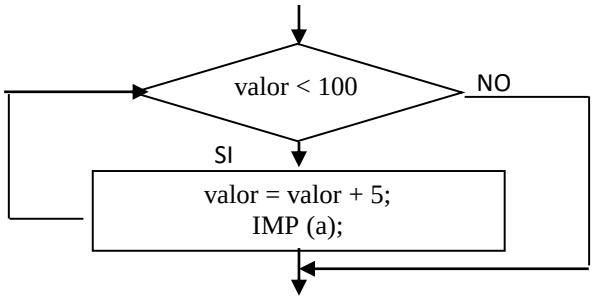
Estructures de repetició

Les estructures de repetició (o bucles) permeten repetir l'execució d'un conjunt de instruccions fins a aconseguir que es compleixi una determinada expressió lògica o condició (la condició per donar per finalitzat el cicle de repeticions).

Tenim els següents tipus d'estructures de repetició:

Estructura de repetició While-Do: Cal avaluar una expressió lògica o condició per entrar a la estructura i executar el bloc d'instruccions que conté (per tant el numero de repeticions del conjunt d'instruccions pot anar de 0 a infinit). Si la expressió lògica o la condició es compleix (dona vertader o un "1") entrarem a dins de la estructura tantes vegades com sigui necessari fins que la condició deixi de complir-se.

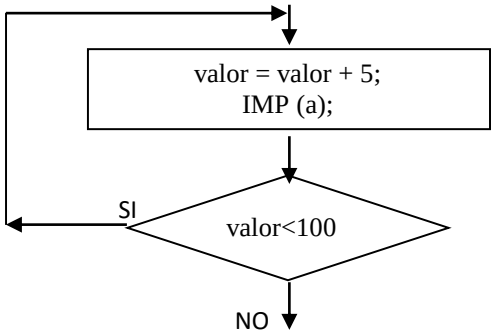
Representació amb un ordinograma:




```
while ($valor < 100) {
  $valor = $valor + 5;
  echo $valor . "<br>;
}
```

Estructura de repetició Do-While: Repeteix un conjunt de instruccions mentre una expressió lògica o condició es compleixi (dona vertader o un "1"), condició que s'avaluarà després de l'execució de les instruccions, per tant el numero de repeticions pot anar des de 1 fins a infinit.

Representació amb un ordinograma:




```
do {
  $valor = $valor + 5;
  echo $valor . "<br>;
} while ($valor < 100);
```

Estructura de repetició For: deriva d'una estructura de tipus While-Do que aprofita la primera línia (on s'avalua l'expressió lògica o condició) per poder incloure opcionalment la inicialització de les variables que farem servir a dins de la estructura (separant-les per comes) i com s'incrementen abans de tornar a avaluar la condició.

```
for ($a=1; $a < 10; $a++)
{
   echo  "<br> $a";
}
```

Explicació: quan entrem a la estructura *$a* val 1, com que es cert que és menor que 10, entrarem dins del bucle a executar les instruccions, al finalitzar s'incrementaran las variables tal i com indiquem al final de la primera línia i es torna a avaluar la expressió. Quan el valor de *$a* sigui igual o major que 10 es sortirà de l'estructura de repetició.

Us dels valors de les variables per sortir de les estructures de repetició:

Per comptador: utilitzant una variable que incrementa o disminueix el seu valor (normalment un sencer), i que al arribar a un valor determinat provoca la sortida del bucle.

Per "sentinella": utilitzant una variable de tipus bandera (flag) que acostuma a ser de tipus boolean (0 o 1, fals o vertader) i que quan canvia de valor provoca la sortida del bucle.

Estructura de repetició foreach: estructura més moderna derivada del for que ens ajuda tant a recórrer conjunts d'elements sense haver de conèixer la seva mida com a recórrer conjunts que no tenen un índex numèric sinó associat a un valor (conjunts o **arrays associatius** que estudiarem més endavant).

```
$matriu = [0, 1, 10, 100, 1000];

foreach ($matriu as $valor) {
   echo "<br>$valor";
}
```

Explicació: per a conjunts clàssics la variable *$valor* va rebent els valors que conté el vector *$matriu* amb l'ordre establert per la seva posició, sense que tinguem que fer ús explícit d'un índex ni conèixer quants elements guarda *$matriu*.

Com ja veurem, **foreach** és especialment útil amb arrays associatius on treballem amb parelles clau-valor .

Operadors als llenguatges de programació.

Els operadors ens permeten realitzar operacions amb les variables a través de les expressions. Una expressió és la representació d'una determinada operació amb els valors de les variables, i necessita dels operadors proporcionats pel sistema.

Exemple d'una expressió, on els operadors són els símbols específics per representar operacions aritmètiques com el igual, els parèntesis, el signe de la suma i el de la multiplicació:

```
$x = ($a + $b)*$c;
```

Tipus d'operadors :

A) Operadors aritmètics: realitzen les operacions matemàtiques bàsiques i els seus símbols son: **+**, **-**, *****, **/**, i el **%** (operador **mod** o residu de la divisió de números sencers)::

```
// imaginem que $x té com a valor inicial 7
    $x = $x + 5;    // equival a:   $x + = 5; ($x ara val 12
    $x = $x – 3;    // equival a:   $x - = 3;  ($x ara val 4)
    $x = $x * 3;    // equival a:   $x * = 3;  ($x ara val 21)
    $x = $x / 2;    // equival a:   $x / = 2;   ($x ara val 3)
    $x = $x % 2;    // equival a:   $x % = 2; ($x  ara val 1)
```

B) Operadors d'increment i decrement unitari: són els operadors especials **++** i **--,** que permeten realitzen aquesta operació de forma abreviada:

```
    $x++;          // equival a:   $x = $x + 1;
    $x--;          // equival a:   $x = $x - 1;
```

Atenció a aquest cas: `++$x;` primer incrementa $x en una unitat i després utilitzarà el valor de $x si forma part d'una expressió. I això és diferent de `$x++;` on s'utilitza primer el valor de $x a dins de la expressió i a continuació li incrementarà el valor en una unitat.

Exemple:

```
$a = 3;
$b = 5
$x = 3;

$result = ( $a + $b ) * $x++;          // $result val 24
$result = ++$x * ( $a + $b );          // $result val 32
```

C) Operadors relacionals i lògics: Els operadors **relacionals** són: **>**, **>=**, **<**, **<=**, **==** (el igual per a comparar valors que és diferent al **=**, que s'utilitza per assignar valors), **===** (operador **idèntic**, les variables hauran de ser iguals en valor i tipus) **!=** ó **<>** (indiquen que els valors son diferents, no miren el tipus).

Els operadors **lògics** són: **&&** (equival al concepte lògic **and**), **||** (equival al concepte lògic **or**), **!** (equival al concepte lògic **not** o negació).

Conceptes AND, OR i NOT:

AND: El resultat serà 1 o vertader si totes les condicions a avaluar valen 1 o són vertaderes, en cas de que una sola ja no ho sigui el resultat serà 0 o fals.

OR: El resultat serà 1 o vertader si una de les condicions a avaluar val 1 o és vertadera, només en cas de que cap de les condicions sigui vertadera el resultat serà 0 o fals.

NOT: El resultat surt de canvia el valor de la condició (negació) de vertader a fals o de fals a vertader en funció del valor inicial de la condició

D) L'operador condicional ? : És una combinació d'operadors que equival a la sentència de control de programa **if-then-else,** amb la limitació de només poder executar una sentencia o instrucció tant en cas de que s'acompleixi la condició a avaluar com al cas contrari.

Exemple:

```
( $a > $b ) ?  $major = $a : $major = $b;
```

// equival al següent codi:

```
if ( $a > $b ) {
        $major = $a;
} else {
        $major = $b;
}
```

E) L'operador de concatenació . Permet unir en una cadena de text els valors de les variables sigui quin sigui el seu format original.

Exemple: Si *$num1=11* y *$num2=22*, utilitzar el operador com *$num1 . $num2* retorna una cadena amb el valor *"1122"* $result = $num1 . $num2; *//1122*

Comentaris al codi

Els comentaris al codi ens ajuden a entendre com funciona el programa (aquesta informació va dirigida al programador, no a l'usuari final). Pot ser important introduir comentaris al codi font dels programes a la fase de desenvolupament, però una vegada el programa estigui preparat per pujar a producció hauríem de treure tots els comentaris.

La idea principal dels comentaris al codi es ajudar-nos a fixar unes bases y establir unes decisions, una vegada ja desenvolupem en aquesta línia la documentació serà sobre un document tècnic extern i no sobre el codi. De fet, si el nom de les variables i les funcions és el adient hauríem de poder llegir el nostre codi igual que un llibre.

Els comentaris a PHP poden ser d'una sola línia o un conjunt de línies:

```
// comentari en una línia
# comentari en una línia
/*
        Comentari extens
        Que pot ocupar
        Tantes línies com sigui necessari
*/
```

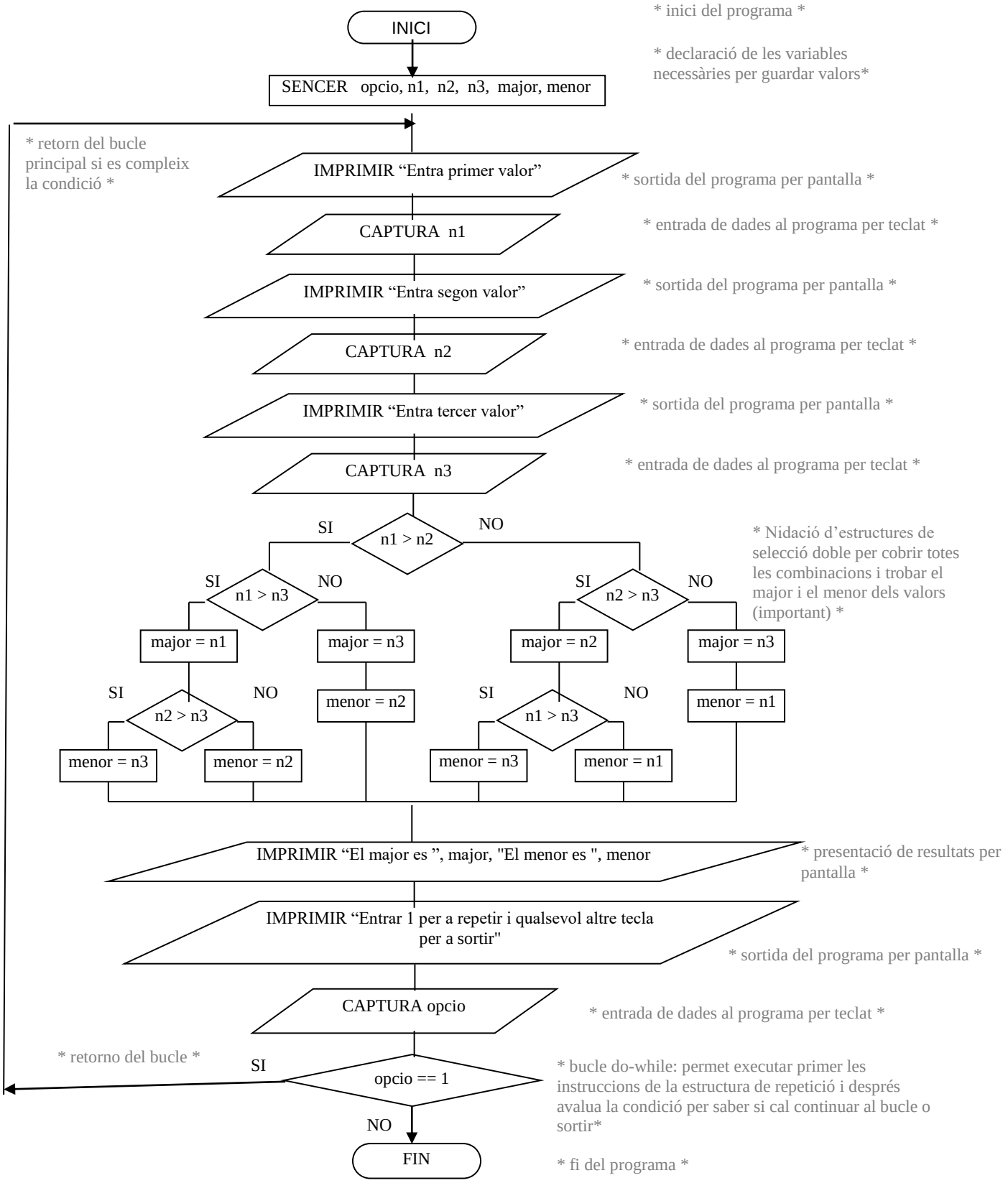
Depuració d'errors

Quan comencem a codificar el programa apareixeran una sèrie d'errors que amb atenció i experiència haurem de depurar fins a obtenir un codi lliure d'errors. El errors comesos poden ser de diferents tipus i es classifiquen segons el moment en que es produeixen o el culpable dels mateixos. Així trobem la següent classificació de tipus d'errors:

- **Errors de Codificació:** són propiciats per absència d'algun símbol específic com els parèntesis, el punt i coma, els claudàtors, alguna estructura de programació malament finalitzada, etc.
- **Errors Màquina:** Són detectats a la memòria principal, la CPU o a qualsevol element que relacioni aquests dispositius. Per tant són errors provocats pel hardware i haurem de depurar si és per una incorrecta gestió del mateix per part del nostre programa.
- **Errors de Programa:** Són deguts a la lògica o a les dades del nostre programa. Exemples: divisió per 0, bucles infinits, assignació incorrecta de tipus de variables, etc.
- **Errors d'Adreçament:** es produeixen quan una instrucció fa referència a alguna variable que no existeix. Exemples: intentar accedir a una variable a través d'un apuntador a una adreça de memòria on realment no es troba la variable.
- **Errors d'Instrucció Privilegiada:** es produeixen quan intentem executar instruccions que necessiten permisos de administradors o usuaris del sistema amb privilegis especials, i l'usuari que executa el programa no disposa d'aquests permisos.

Exemples Disseny d'algorismes

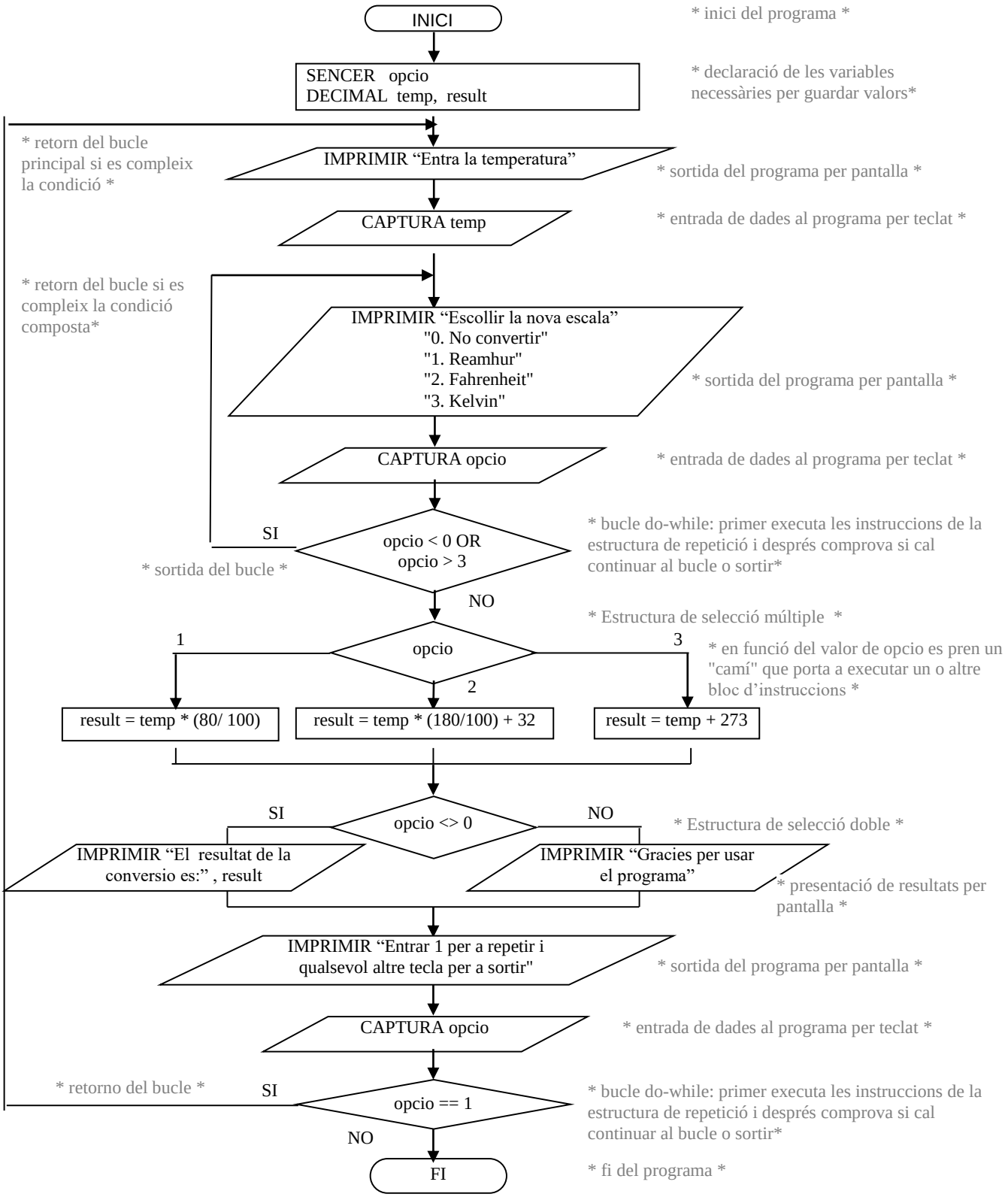
Dissenyar l'ordinograma d'un programa que rep per teclat 3 valors numèrics sencers i presentar per pantalla un missatge indicant quin és el valor més alt i quin el més baix dels 3. El programa donarà l'oportunitat a l'usuari de repetir la seva execució per provar diferents combinacions.



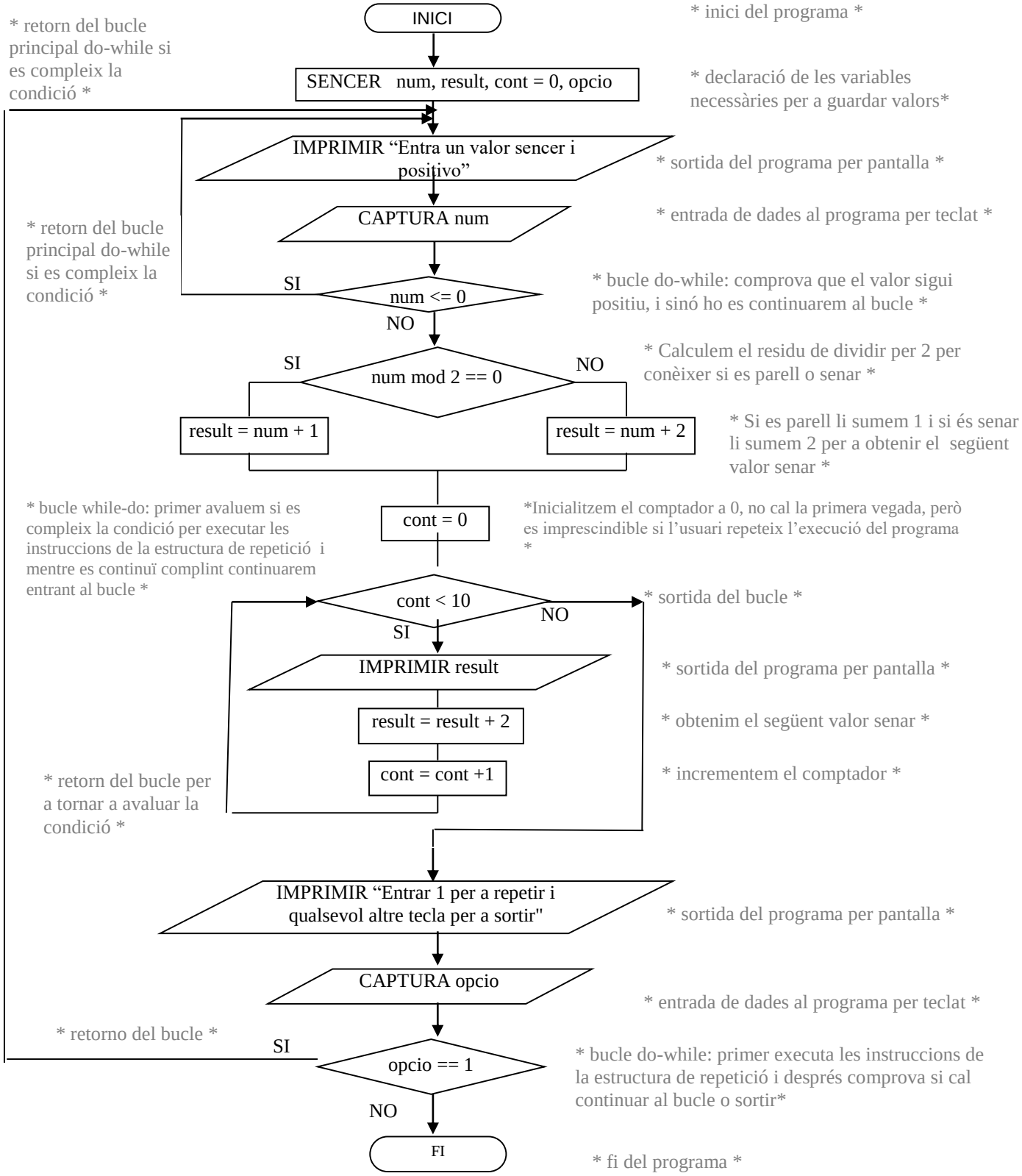
Dissenyar l'ordinograma d'un programa que rep per teclat el valor d'una temperatura en graus centígrads (valor numèric que pot tenir decimals) i amb un menú es pot escollir la seva conversió a les escales Reamhur, Fahrenheit o Kelvin. El programa permetrà a l'usuari repetir la seva execució per provar amb diversos valors.

Per a calcular les formules de conversió es presenten les següents equivalències:

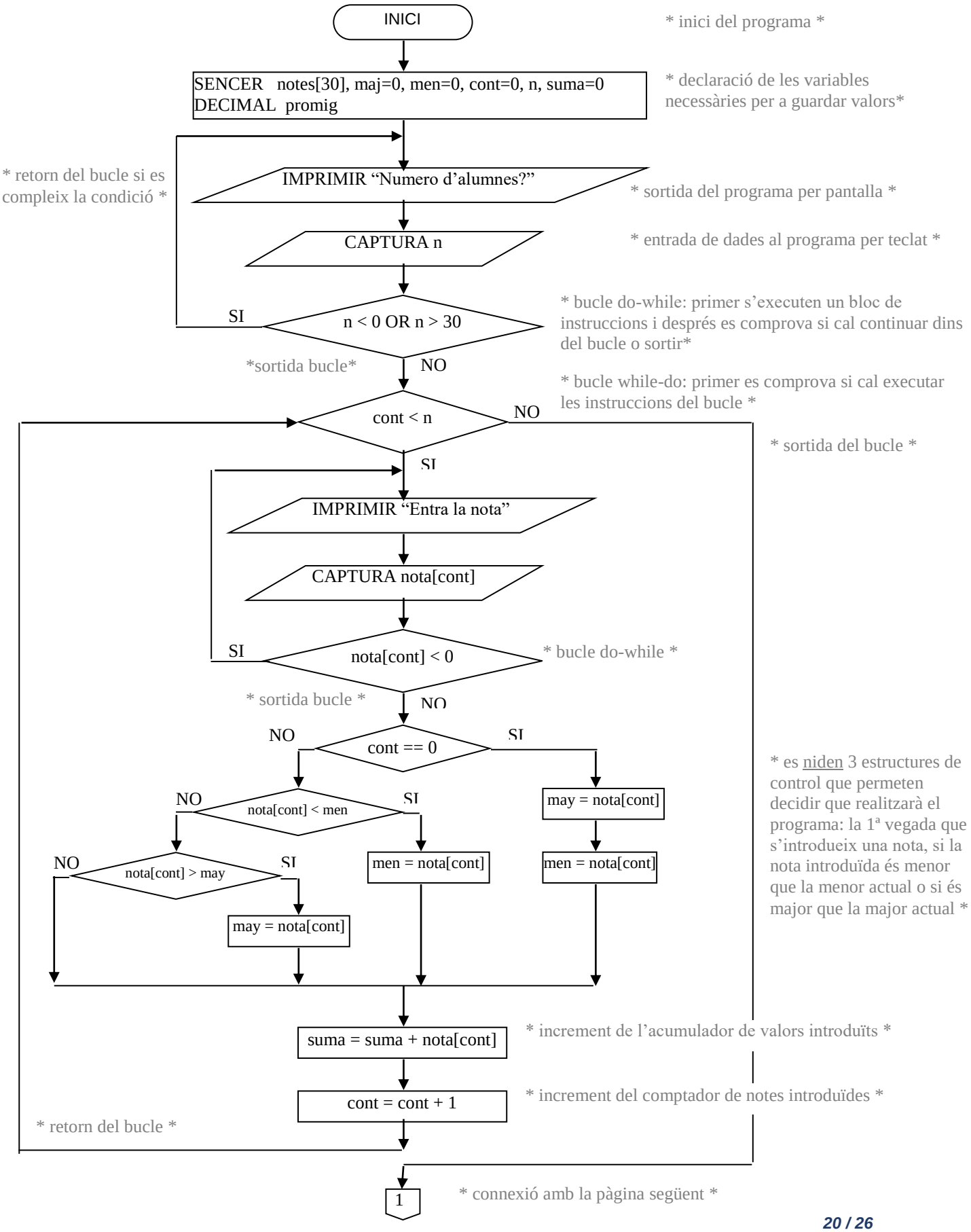
100 º centígrads = 80 º Reamhur = 212 º Fahrenheit = 373 º Kelvin
0 º centígrads = 0 º Reamhur = 32 º Fahrenheit = 273 º Kelvin

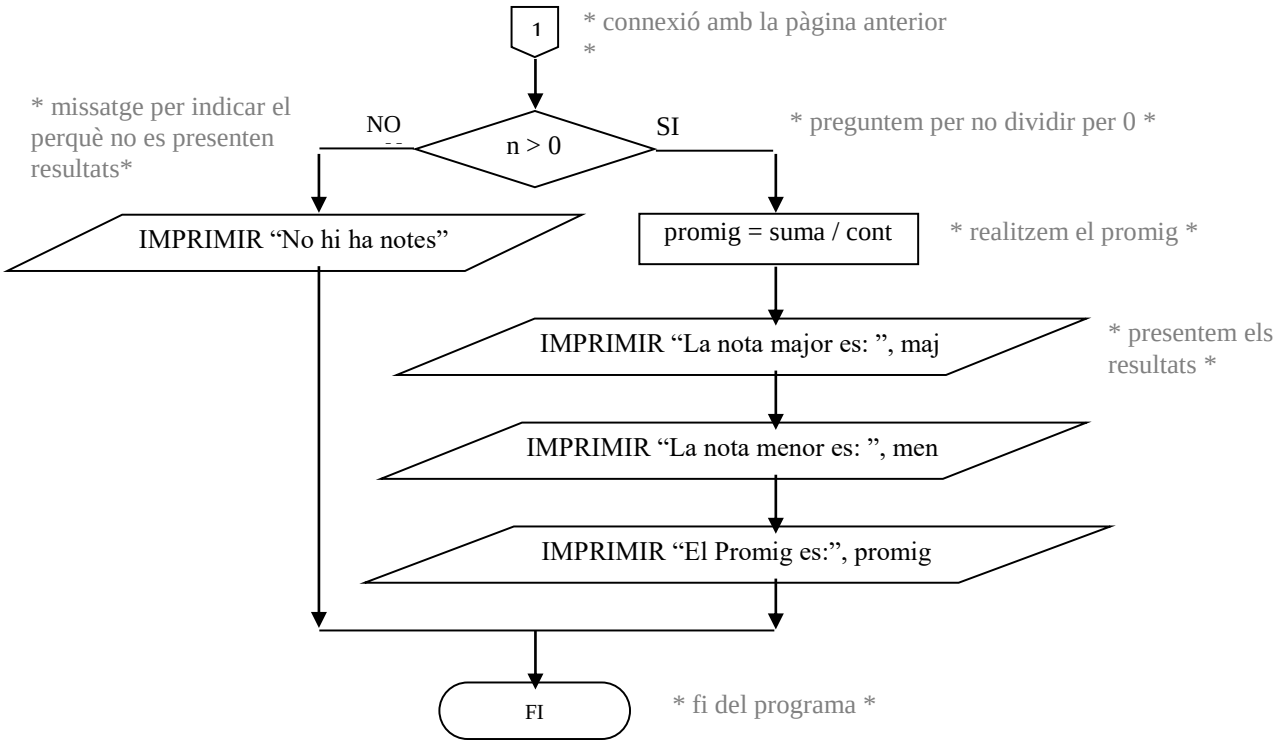


Dissenyar el ordinograma d'un programa que rep un número sencer i positivo per teclat e imprimeix els 10 primers números senars majors que el número rebut. El programa donarà l'oportunitat a l'usuari de repetir la seva execució per provar amb diversos valors.



Dissenyar l'ordinograma d'un programa per capturar les notes dels exàmens de N alumnes (N serà un numero sencer i positiu més petit de 30 entrat per teclat), i presentar quina ha estat la nota més alta, la més baixa i la nota promig del grup.




1
* connexió amb la pàgina anterior *
* missatge per indicar el perquè no es presenten resultats*
NO
n > 0
SI
* preguntem per no dividir per 0 *
IMPRIMIR “No hi ha notes”
promig = suma / cont
* realitzem el promig *
IMPRIMIR “La nota major es: ”, maj
* presentem els resultats *
IMPRIMIR “La nota menor es: ”, men
IMPRIMIR “El Promig es:”, promig
FI
* fi del programa *

Funcions matemàtiques pel tractament de valors numèrics

El valor numèrics precisen d'un tractament específic, normalment amb funcions que recolzen proves o valors típics en combinatòria, estadística, trigonometria o casos tan habituals com generar valors aleatoris.

A la documentació oficial trobem el llistat complert de funcions per consultar-lo quan calgui:

https://www.php.net/manual/es/ref.math.php

Aquí trobem funcions com **rand()** que ens permet generar valors aleatoris:

```
$valor = rand();                    // de 0 al màxim dels sencers
$valor = rand(1000, 1000000);       //entre els 2 valors, inclosos
```

O funcions que ens calculin l'arrel quadrada d'un valor:

```
echo "<br>La arrel quadrada de $valor es " . (int)sqrt($valor);
```

Mostrant com arrodonir el resultat de la arrel a un valor sencer, si cal.

Funcions pel tractament de cadenes de text

Les cadenes de caràcters necessiten d'un tractament específic pel significat propi que té un conjunt de caràcters (des d'una paraula o frase intel·ligible per a un esser humà a un comandament del sistema amb un conjunt de paràmetres...). A la biblioteca de funcions de PHP trobem un ampli ventall de possibilitats.

https://www.php.net/manual/es/ref.strings.php

Aquestes funcions es poden complementar amb altres famílies de funcions que treballen sobre strings com les expressions regulars. Encara que amb els anys les acabarem coneixent pràcticament totes, hauríem començar per alguns exemples típics d'us, classificats pel tipus d'acció a realitzar sobre el string:

Presentació del valor d'un string o algun subconjunt dels seus caràcters:

- chr — retorna un carácter específico
- echo — Muestra una o más cadenas
- ord — devuelve el valor ASCII de un caracter

- print — Mostrar una cadena
- printf — Imprimir una cadena con formato
- sprintf — Devuelve un string formateado

Mesurar longitud del string o comptar nombre d'una ocurrència:

- count_chars — Devuelve información sobre los caracteres usados en una cadena
- str_word_count — Devuelve información sobre las palabras utilizadas en un string
- strlen — Obtiene la longitud de un string
- substr_count — Cuenta el número de apariciones del substring

Conversió majúscules <-> minúscules:

- lcfirst — Pasa a minúscula el primer caracter de un string
- strtolower — Convierte una cadena a minúsculas
- strtoupper — Convierte un string a mayúsculas
- ucfirst — Convierte el primer caracter de una cadena a mayúsculas
- ucwords — Convierte a mayúsculas el primer caracter de cada palabra de una cadena

Comparació:

- strcmp — Comparación de string segura a nivel binario
- strcasecmp — Comparación de string segura a nivel binario e insensible a mayúsculas y minúsculas
- strncmp — Comparación segura a nivel binario de los primeros n caracteres entre strings
- strncasecmp — Comparación de los primeros n caracteres de cadenas, segura con material binario e insensible a mayúsculas y minúsculas
- substr_compare — Comparación segura a nivel binario de dos o más strings desde un índice hasta una longitud de caracteres dada

Localització de substrings:

- strstr — Encuentra la primera aparición de un string
- strchr — Alias de strstr
- stristr — strstr insensible a mayúsculas y minúsculas
- stripos — Encuentra la posición de la primera aparición de un substring en un string sin considerar mayúsculas ni minúsculas
- strpos — Encuentra la posición de la primera ocurrencia de un substring en un string
- strripos — Encuentra la posición de la última aparición de un substring insensible a mayúsculas y minúsculas en un string
- strrchr — Encuentra la última aparición de un caracter en un string
- strrpos — Encuentra la posición de la última aparición de un substring en un string

Extracció de substrings:

- chunk_split — Divide una cadena en trozos más pequeños
- explode — Divide un string en varios string
- sscanf — Interpreta un string de entrada de acuerdo con un formato
- str_split — Convierte un string en un array
- str_getcsv — Convierte un string con formato CSV a un array
- substr — Devuelve parte de una cadena

Altres transformacions i/o substitucions:

- trim — Elimina espacio en blanco (u otro tipo de caracteres) del inicio y el final de la cadena
- chop — Alias de rtrim
- ltrim — Retira espacios en blanco (u otros caracteres) del inicio de un string
- rtrim — Retira los espacios en blanco (u otros caracteres) del final de un string
- implode — Une elementos de un array en un string
- join — Alias de implode
- str_replace — Reemplaza todas las apariciones del string buscado con el string de reemplazo
- str_ireplace — Versión insensible a mayúsculas y minúsculas de str_replace
- substr_replace — Reemplaza el texto dentro de una porción de un string

- str_pad — Rellena un string hasta una longitud determinada con otro string
- str_repeat — Repite un string
- str_shuffle — Reordena aleatoriamente una cadena
- strrev — Invierte una string
- strip_tags — Retira las etiquetas HTML y PHP de un string
- number_format — Formatear un número con los millares agrupados

Codificació:

- crypt — Hash de cadenas de un sólo sentido
- md5_file — Calcula el resumen criptográfico md5 de un archivo dado
- md5 — Calcula el 'hash' md5 de un string
- sha1_file — Calcula el hash sha1 de un archivo
- sha1 — Calcula el 'hash' sha1 de un string

Dades compostes

Fins ara hem treballat amb un tipus de dada molt simple, ja sigui nombre sencer o amb decimals, o de tipus string. Però quan avancem a la complexitat dels problemes necessitarem dades més complexes, que permetin agrupar les diferents característiques d'un tipus d'informació. Per exemple dades de capitals europees.

Per treballar amb dades compostes fem servir els **arrays associatius** on podem guardar conjunts de dades "clau-valor". La clau identifica el tipus de dada i el valor el seu contingut. Podem definir variables simples o arrays d'aquest tipus de dada composta.

Exemple d'us de l'array associatiu per guardar una dada composta:

$dada = ["pais"=>"España", "capital"=>"Madrid", "poblacio"=>"4 milions"],

Per accedir als seus valors cal indicar el nom de la variable i el nom de la clau:

echo $dada ["capital"];

Podríem fer servir arrays no associatius excepcionalment per guardar dades de diversos tipus al vector, PHP ho permet però no es una forma massa neta de treballar:

```
$conjunto = ["vermell", "verd", "blau", 6, true, 10.4];
```

I podríem seguir afegint valors de tot tipus, però no es una bona pràctica

```
$conjunto[] = "groc";
```

Exemple d'us de l'array associatiu per guardar conjunts de dades compostes:

```
$capitalsEuropa = [
                        ["pais"=>"España", "capital"=>"Madrid", "poblacio"=>"4 milions "],
                        ["pais"=>"Francia", "capital"=>"Paris", "poblacio"=>"6 milions "],
                        ["pais"=>"Italia", "capital"=>"Roma", "poblacio"=>"5 milions "]
];

echo "<br> Estat: " . $capitalsEuropa[0]["pais"];
echo "<br> Ciutat: " . $capitalsEuropa[0]["capital"];
echo "<br> Habitants: " . $capitalsEuropa[0]["població"];
```

NOTA: només podem guardar o mostrar les dades d'una variable composta, camp a camp.

Amb aquest tipus de variables té encara més sentit fer servir el bucle **foreach**, però amb la següent sintaxi que encara el fa més interessant:

```
foreach ($matriu as $index => $valor) {

}
```

On podrem recórrer la matriu de dades, capturant tant la clau que ara de índex a la variable $index com el seu contingut a la variable $valor.

Un exemple de la seva flexibilitat amb el conjunt de dades compostes anterior:

```
foreach ($capitalsEuropa as $dada) {

        foreach ($dada as $index => $valor){
                echo "<br>El camp $index conte $valor";
        }
}
```

Una manera simple de resoldre un recorregut difícil de realitzar sense aquest tipus de estructura de repetició.