Nota introductòria:

Aquest treball va orientat a la gent que no te ni idea de programar i vol aprendre'n. Us servirà per fer−vos una idea de com és la programació, en general, també us explica els principals principis i eines de treball d'un programador, així com els principals tipus de dades que s'utilitzen per fer programes.

El més important és que crec que us ajudarà a entendre altres manuals incomprensibles que us ensenyin a programar de veritat.

ÍNDEX

- INTRODUCCIÓ A LA PROGRAMACIÓ
- **Concepte de programa i llenguatge de programació.**

Els ordinadors personals estan formats per dues parts ben diferenciades, l'hardware (CPU i els dispositius perifèrics) i el *software* (sistema operatiu i programes). I les definim:

Hardware: Tot el conjunt de components electrònics i mecànics o físics, d'una computadora (*microprocessador*, *RAM*,* impressora...)

Software: Conjunt de programes o procediments que determinen les accions de les màquines (serveix per controlar l'hardware).

Els ordinadors es poden considerar sistemes informàtics, sabent que sistema és un conjunt de components relacionats que interactuen per a realitzar una tasca. Un PC està estructurat, segons el model que es pot observar a la figura 1, que descriu les interaccions que es realitzen entre els elements que integren un ordinador.

FIGURA 1 *Esquema de les interaccions entre els components d'un ordinador.*

Així, si volem imprimir un text, visitar una pàgina web, visualitzar en el monitor el que estem escrivint etc., hem d'indicar a la impressora que imprimeixi el que hi ha a la pantalla, hem de dir−li al mòdem que ens connecti amb Internet, també hem d'ordenar al monitor que visualitzi lletra per lletra el que estem escrivint etc. Tot això és *programar*, per tant podem deduir que la programació es basa en les relacions que es mantinguin entre una màquina i el seu hardware, ordenant a aquest, una sèrie d'instruccions a partir de les quals s'ha de dur a terme una acció desitjada pel programador. Aquestes instruccions (imprimeix el text, reprodueix aquesta cançó, connectat a Internet etc.) , es codifiquen en un llenguatge comprensiu per la màquina.

Aleshores podem dir que un *programa* és una sèrie d'instruccions donades a l'ordinador, en un llenguatge comprès per ell, per dir−li exactament el que volem fer. Tots els llenguatges que ens serveixen per a escriure programes reben el nom de *llenguatges de programació*.

- **Història, orígens i evolució de la programació**

Els primers ordinadors es programaven i acceptaven dades a partir de la modificació dels seus circuits o l'alteració del seu hardware, la qual cosa feia que el procés fos lent i molt costós i no oferia gaires prestacions. Amb el pas del temps, el volum de dades a processar va

anar en augment fins al punt que no va quedar més remei que desenvolupar nous sistemes, com van ser les targetes o *cintes perforades.*

Amb aquest nou sistema, les dades de cada programa podien emmagatzemar−se en aquestes cintes, formant una successió de forats, això va comportar que els programes poguessin carregar−se, de manera més fàcil i ràpida, a la memòria de l'ordinador. Tot i l'enorme avanç que van suposar les cintes perforades, encara era un procediment lent i laboriós. A arrel d'això va sorgir la necessitat de trobar una manera més fàcil i senzilla de comunicar−se amb l'ordinador i codificar els programes, el que va originar l'apacrició dels

llenguatges de programació.

L'evolució de la programació sempre ha seguit paral·lelament a la dels ordinadors. Els primers llenguatges de programació intentaven aprofitar al màxim la poca capacitat dels processadors.

En aquesta època era freqüent la programació que es realitzava mitjançant l'ús d'un *llenguatge de baix nivell* o *ensamblador*, que substituïa al *llenguatge màquina** dels ordinadors, el qual consisteix en un codi format únicament pels símbols 0 i 1, el codi binari. L'ensamblador va servir, principalment per a fer més legible els

programes, però també introduïa instruccions addicionals que no corresponien a cap instrucció per a la màquina, sinó que eren per al *compilador**. Aquestes instruccions indiquen l'inici i el final de diferents seccions dels programes i unes altres de més concretes, s'anomenen *pseudoinstruccions*. Però aviat es va comprovar que aquests llenguatges no eren la solució definitiva al problema de la programació, ja que en canviar de processador era necessari un nou ensamblador adaptat a la nova màquina, a més, per a programar amb ensamblador s'ha de conèixer a fons el microprocessador, estructura de la memòria i moltes més característiques.

A pesar d'aquests inconvenients, els llenguatges simbòlics o ensambladors se segueixen utilitzant en l'actualitat, degut principalment, a què existeix una enorme quantitat de codi escrit en llenguatges d'aquest tipus, de la que no es pot prescindir, i encara són necessaris per a programar certes seccions crítiques dels programes ja que aquests llenguatges, en ser tant propers al codi màquina, permeten realitzar programes especialment adaptats a ella i que funcionen molt ràpidament.

El següent pas en aquest camp va ser la creació dels *llenguatges d'alt nivell*, que eren més independents de la màquina i de molt més fàcil comprensió, per la qual cosa qualsevol usuari amb coneixement d'un d'aquests llenguatges pot programar qualsevol ordinador, sempre que aquest disposi del compilador o *intèrpret** del llenguatge que vol utilitzar. Aquests llenguatges, treballen en dues fases: la primera consisteix a traduir el codi utilitzat per l'usuari (o codi font) a un codi intermedi i en la segona fase tradueixen aquest codi intermedi (ensamblador) al llenguatge específic de la màquina en la que es desitja executar el programa. D'aquesta manera, modificant únicament el programa encarregat de la segona fase, s'aconsegueix que un programa es pugui traduir al llenguatge de màquines diferents amb molt poc esforç.

1ª fase 2ª fase

FIGURA 3 *Traducció d'un llenguatge d'alt nivell al llenguatge màquina.*

A mesura que apareixien els llenguatges d'alt nivell al mercat, els usuaris han anat comprovant que alguns eren més adequats que altres per a la realització de diferents tasques. Per exemple el *Cobol*, un dels primers llenguatges d'alt nivell, es va convertir en el més apreciat per a la creació d'aplicacions de gestió i el *Fortran*, també un dels primers en aparèixer, era preferible als altres per treballar amb fórmules i càlculs més complexos.

Actualment, s'utilitzen més habitualment llenguatges d'alt nivell més elaborats i que simplifiquen molt la programació, com el *Pascal*, el *C i C++* i el *Java* que està entrant amb molta força, està especialment pensat per Internet i també permet crear aplicacions força complexes.

Tot i així, anava en augment la necessitat de crear programes complexes d'una forma més ràpida i senzilla. A causa d'això, aparegueren al mercat els *llenguatges de programació visual* que treballen sobre la base de llenguatges d'alt nivell, com *Basic*, Pascal o C++, i permeten crear programes a partir de la selecció d'objectes que es vol que apareguin en la pantalla (Per això també reben el nom de *llenguatges orientats a objectes)*. Gràcies a això els usuaris poden fer programes sense la necessitat de tenir grans coneixements de sintaxis complexes. Uns d'aquests llenguatges son, *Delphi*, de la casa Borland i *Visual Basic*, de Microsoft.

- **Classificació dels llenguatges de programació**

Els llenguatges de programació es classifiquen segons dos criteris.

El primer, segons la facilitat o dificultat del microprocessador de processar el llenguatge, és a dir, descodificar−lo i traduir−lo al llenguatge màquina. El segon criteri, els classifica per la seva evolució.

Segons el primer criteri, els classifiquem en nivells i segons el segon criteri els classifiquem en generacions.

- ***Els nivells dels llenguatges de programació.***

Els llenguatges es classifiquen en tres nivells principals: baix, intermedi i alt.

El *llenguatge de baix nivell* és el llenguatge màquina, que és l'únic que comprèn directament la computadora, com hem dit abans, utilitza un codi format únicament per dos símbols, 0 i 1, denominats *bits* (abreviatura anglesa de dígit binari), va ser el primer llenguatge utilitzat en la programació d'ordinadors. Lògicament, en estar format per dos tipus de caràcters, és molt difícil i complicat d'entendre pels programadors i va ser substituït per altres, més fàcils d'utilitzar (llenguatges d'alt nivell i intermedis), reduint, d'aquesta manera la possibilitat de cometre errors.

Els *llenguatges intermedis* també reben el nom de *llenguatge ensamblador*. *A*quest llenguatge és el primer intent de substituir el llenguatge màquina per un altre, més senzill i semblant a l'utilitzat per les persones. En aquest llenguatge cada instrucció equival a una instrucció del llenguatge màquina, utilitzant símbols alfanumèrics, de tal manera que les instruccions quedaven força abreviades i era més comprensible pel programador. Encara que presentava una sèrie d'inconvenients propis del llenguatge màquina i eren força notables:

- Cada model d'ordinador té un llenguatge ensamblador propi, per la qual cosa l'ús d'un programa quedava limitat als ordinadors d'un mateix tipus.
- El programador ha de conèixer perfectament l'hardware de l'equip, ja que manipula directament la memòria, registres del processador i altres elements físics.
- Totes les instruccions són elementals, és a dir, s'han de descriure detalladament totes les operacions que s'ha de dur a terme en la màquina per a la realització de qualsevol procés.

Per una altra banda, tant el llenguatge màquina com l'ensamblador tenen l'avantatge de la mínima ocupació de memòria i mínim temps d'execució en comparació amb el resultat de la compilació del mateix programa en un altre llenguatge (d'alt nivell). Per això, degut a aquestes semblances entre els dos llenguatges, en moltes ocasions s'inclou el llenguatge ensamblador en el grup de baix nivell.

Els *llenguatges d'alt nivell* o també anomenats llenguatges evolucionats, constitueixen un grup molt més complex, el formen la resta de llenguatges de programació. Els caracteritza l'ús de compiladors o traductors, ja que han de traduir les ordres d'aquests llenguatges a llenguatge màquina. Aquests llenguatges van sorgir, principalment, amb els següents objectius,:

- Aconseguir independència de la màquina, podent utilitzar un mateix programa en diferents tipus d'ordinadors amb l'única condició de disposar d'un compilador o traductor, esmentats anteriorment. Gràcies a això no és necessari conèixer l'hardware específic de la màquina.
- Aproximar−se al llenguatge humà, perquè el programa es pugui escriure i llegir d'una forma més senzilla, eliminant moltes més possibilitats de cometre errors que es donen en el llenguatge màquina o en l'ensamblador ja que s'utilitzen paraules (en anglès) en comptes de cadenes de signes sense cap significat arparen.
- Incloure *rutines* o *procediments* d'ús freqüent com les d'entrada i sortida, funcions matemàtiques, manipulació de taules, etc, de tal manera que es poden utilitzar sempre que es vulgui sense necessitat de programar−les cada vegada.

El principal problema dels llenguatges d'alt nivell és el gran nombre d'aquests, a part de les diferents versions o dialectes que s'han desenvolupat d'alguns d'ells. Uns quants exemples de llenguatges d'aquest tipus son: Fortran, Lisp, Algol, Cobol, Apl, Snobol, Prolog, Modula2, C/C++, Java, Pascal, Basic i Matlab.

- ***Les generacions dels llenguatges de programació***

També podem classificar els llenguatges de programació en cinc generacions ben diferenciades, encara que algunes vegades se'n consideren únicament quatre. Aquesta classificació és deguda a la necessitat de disposar d'eines de programació per treure el màxim profit dels ordinadors a mesura que aquests evolucionaven. Les generacions són les següents.

Primera generació: Constituïda pel llenguatge màquina, explicat anteriorment.

Segona generació: La formen els llenguatges ensambladors que, com hem dit abans, depenen de cada model d'ordinador.

Tercera generació: El següent pas va ser la creació de llenguatges d'alt nivell, que són els que constitueixen aquesta generació.

Quarta generació: Aquesta generació correspon a una sèrie d'eines que permeten construir aplicacions senzilles combinant segments de programa ja fabricat. En algunes ocasions es creu que aquestes eines no són un llenguatge sinó extensions de llenguatges ja existents. Els llenguatges que integren aquesta generació reben són els llenguatges propis de la programació orientada a objectes (POO).

Cinquena generació: S'inclouen en aquest grup els llenguatges utilitzats per crear programes que utilitzen la *intel·ligència artificial** (IA). Encara que, com ja hem anunciat, aquesta denominació no s'utilitza gaire.

- ASPECTES BÀSICS DE LA PROGRAMACIÓ
- **Algorisme**

Un algorisme és una seqüència d'accions que, executades en un determinat ordre, soluciona un determinat problema.

Les seves característiques fonamentals són:

- Ha de ser precís i indicar l'ordre de realització de cada pas.
- Ha d'estar definit (si es repeteixen *n* vegades les passes s'ha d'obtenir sempre el mateix resultat).
- El nombre de passes ha de ser finit.
- És independent del llenguatge de programació que s'utilitzi.

Els algorismes són fonamentals en la programació ja que per fer programes mínimament complexes és molt convenient escriure'ls, prèviament, a mà (sense utilitzar el codi del programa). D'aquesta manera, és molt més fàcil i ràpid pensar el programa i després traduir−lo al llenguatge utilitzat que no posar−nos a picar el codi i alhora pensar quins elements o quines ordres hem d'utilitzar, perquè el programa faci el que nosaltres desitgem.

A continuació (en la figura 4) podrem veure un exemple d'un algorisme i la posterior traducció al codi del llenguatge (Pascal).

Suposem que volem aconseguir la suma de dos nombres enters:

Algorisme

INICI

Llegir el valor A (enter).

Llegir el valor B (enter).

Obtenim C (enter) com a suma de A i B.

Mostrem el valor C a la pantalla.

FINAL

Codi en Pascal

Var

A, B, C: Integer;

begin

ClrScr;

Write(`Introdueixi el primer valor');

ReadLn(A);

Write (`Introdueixi el segon valor');

ReadLn(B);

C:= A + B;

WriteLn(`El resultat és =', C)

end

FIGURA 4 *L'algorisme i el codi en Pascal d'un programa que suma dos valors enters.*

Un aspecte molt important dels algoritmes és que, per resoldre un mateix problema existeixen un nombre infinit de solucions, el més important és saber escollir quina solució és la més ràpida i senzilla. Això és un dels principals motius que diferencia els bons programadors dels altres.

- **Disseny d'un algorisme**

Com hem dit abans, els algorismes es basen en la resolució de problemes. Aquesta resolució té tres fases que veurem més detalladament a continuació:

- Anàlisi del problema.
- Disseny de l'algorisme.
- Resolució en la computadora: Traducció de l'algorisme a un llenguatge de programació (mirar figura 4).

- *Anàlisi del problema.*

Consisteix en comprendre el problema i saber clarament què és el que *entra* i què és el que *surt*, és a dir, la informació introduïda per l'usuari i la informació amb la que el programa ha de respondre. També hem de saber quina és l'*operació* que vincula l'entrada amb la sortida, o sigui, què s'ha fet perquè a partir de l'entrada s'obtingui el que surt.

Si ens fixem en l'exemple de la figura 4 i analitzem el problema, obtenim com a entrada dos nombres enters i com a sortida la suma dels valors anteriors i l'operació que els vincula és la suma.

- *El disseny de l'algorisme.*

Consisteix en, havent entès el problema, determinar quines accions hem de realitzar per a resoldre'l.

Un cop tenim la solució al problema, s'ha d'implementar amb alguna representació. Les més utilitzades són els *diagrames de flux*, els *diagrames N−S* i el *pseudocodi*.

El diagrama de flux és una notació gràfica basada en la utilització de símbols, anomenats caixes, en les que escrivim les accions que ha de realitzar l'algorisme, les caixes estan connectades mitjançant línies. Això ens indica l'ordre en el que hem d'executar les accions. Hi ha dos tipus principals de caixes, el rectangle i el rombe, en la primera s'hi escriuen les accions que ha de realitzar la computadora i en la segona s'hi escriuen condicions que determinaran l'execució d'unes accions o d'altres. Hi ha moltes més figures a part del rectangle i el rombe, com l'el·lipse (que serveix únicament per marcar l'inici i el final). Els símbols més utilitzats són els següents:

Símbol de procés: Indica l'acció que ha de realitzar la computadora.

Representa les accions d'entrada i sortida (accions de lectura i escriptura).

Condició: Determina l'execució d'unes accions o unes altres.

Inici i final: Hi escrivim les paraules inici i final de l'algorisme.

Subprograma: Escrivim el subprograma al que cridem.

A part d'aquests símbols n'hi ha d'altres que representen accions de la impressora, del teclat, de la pantalla etc,.

L'altra notació que és una mica semblant a un diagrama de flux, però sense fletxes (les caixes van unides) i amb uns altres símbols és el *diagrama N−S o de Nassi−Schederman*. En aquesta notació les condicions es representen amb triangles i les accions s'escriuen dins d'un rectangle. Encara que aquí sorgeix un altre tipus de caixes, les *repetitives*, que s'utilitzen en els casos en què s'ha de realitzar un conjunt d'accions depenent d'una sola condició, en les caixes repetitives la condició i les accions s'escriuen dins de rectangles però l'acció es troba dins la condició. Aquests són els tipus de caixes:

Condicional 1: Repetitiva 1:

<condició> Des de *a=x1* fins a *xp*

SI NO

<accions>

<acció 1> <acció 2>

Repetitiva 2: Repetitiva 3:

Mentre <condició>

<accions>

<accions> repetir fins a <condició>

Aleshores, l'algorisme de la figura 5, en aquesta notació, quedaria de la següent manera:

FIGURA 6 *Diagrama N−S.*

El pseudocodi, es considera un llenguatge d'especificació d'algoritmes, és molt semblant a qualsevol llenguatge de programació, però té l'avantatge que no es regeix per les normes d'un llenguatge en particular. Diem que és molt semblant a un llenguatge de programació, perquè la traducció pseudocodi−llenguatge és, pràcticament, literal, és a dir, línia per línia. L'algorisme de la figura 4, està escrit amb aquest mètode i podem comprovar la semblança amb el codi del llenguatge al què està traduït.

És important que, sempre que vulguem escriure un algorisme en una de les tres notacions escrivim l'anunciat del programa (problema) abans de fer res.

Normalment s'utilitzen més les dues últimes notacions ja que el diagrama N−S és molt útil alhora de confeccionar l'algorisme i és menys laboriós que l'organigrama. El pseudocodi és molt útil per traduir−lo al llenguatge desitjat.

Algorisme arrel d'un nombre

INICI

Llegir el nombre N.

Si N és positiu o zero

Calcular l'arrel

Escriure el resultat a la pantalla

Si no ho és

Escriure (No es pot calcular l'arrel d'un nombre negatiu)

FINAL

FIGURA 7 *Mètode del pseudocodi*

- **Tipus de dades**

Tots els diferents objectes d'informació amb els que un programa treballa es coneixen com a *dades*. Una dada pot ser un caràcter, com la lletra r, un valor enter, com 35 o un nombre real, com 143,87. Una operació de suma no té sentit amb caràcters, només amb nombres. Conseqüentment, si el compilador detecta una operació

de suma de dos caràcters, produirà un error i deixarà de compilar el programa. Per evitar aquests errors d'operacions i determinar com executar−les s'assigna a cada dada un tipus que haurà de reconèixer el compilador.

Les dades tenen tres característiques:

- Un *nom* que les diferencia de la resta.
- Un *tipus* que ens determina les operacions que podem fer amb elles.
- Un *valor* que pot variar o no en el transcurs de l'operació.

El compilador d'un *llenguatge de tipus fort* detecta molts errors de programació abans que el programa s'executi, d'aquesta manera és menys costosa la depuració del programa ja que trobar aquests tipus d'errors personalment és molt dificultós i fa que perdem molt de temps.

El tipus de cada dada determina la naturalesa del conjunt de valors que pot prendre una *variable**. També és important, en el cas de programes més complexes tenir en compte l'espai de memòria, mesurat amb *bytes**, ocupat per una variable d'un tipus de dada.

Els tipus de dades més importants són:

- *Numèrics*: Engloba tots els nombres separant els *enters* dels *reals*.
- *Lògics o boleans*: Aquells que només poden prendre dos valors (vertader o fals).
- *Caràcters*: Avarca el conjunt finit i ordenat de caràcters que reconeix la computadora (lletres, dígits, caràcters especials...)

Cada llenguatge pot afegir més tipus o també modificacions d'aquests tres, per exemple, Pascal utilitza els tipus *double i real*, els dos són nombres reals però un reconeix més xifres que l'altre i, per tant, la variable d'un ocuparà més memòria que l'altre.

Els tres tipus de dades anomenats anteriorment reben el nom de *simples* ja que les variables d'aquest tipus únicament magatzemera un element. A part d'aquests existeixen uns altres tipus de dades també molt importants, com el *tipus cadena* (una variable d'aquest tipus emmagatzema una cadena de caràcters com pot ser una paraula).

Els tipus de dades més freqüents en els llenguatges de programació són els següents:

Simples	Estàndard	enter
		real
		caràcter
		lògic
	Definides pel programador	subrang
	(No estàndard)	enumerats
estructurades	Simples o estàtiques	arrays (vectors/matrius)
		registres
		arxius
	Compostes o dinàmiques	conjunts
		cadenes
		llistes
		llistes enllaçades

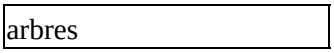

FIGURA 8 *Tipus de dades més freqüents en els llenguatges de programació*

Com ja hem vist, les *dades simples* tenen en comú que cada variable representa un element i en les *estructurades*, un *identificador** pot representar un gran nombre de dades individuals i ens permet referir−nos a cada una d'elles independentment de les altres.

Les *estructures estàtiques* són aquelles en les que la grandària de memòria ocupada es defineix abans que el programa s'executi i no es pot modificar en l'execució. En canvi, en les *dinàmiques* no cal una definició prèvia de la grandària de la memòria.

Tots els tipus de dades de la figura 8 són molt freqüents en un llenguatge de programació, però veurem únicament les més usuals (les simples i algunes de les estructurades), el nom que els hi donen els llenguatges de programació, si és que en tenen un d'específic, i les *variacions* que utilitza Pascal.

- *Tipus de dades simples*:

Estàndard: Són les que venen definides pel llenguatge. O sigui, la seva extensió està predefinida. Per exemple el tipus *integer*, en el llenguatge *Delphi*, té una extensió des de −2147483648 fins a 2147483647. Això vol dir que si, a una variable integer li *assignem* un valor superior al màxim o inferior al mínim, el compilador produirà un error.

Enters: Tots els nombres enters (no accepten part decimal).

Nom: *integer*.

Variacions: *longint, byte, shortint...*

Reals: Tots els nombres reals (accepten part decimal)

Nom: *real*.

Variacions: *single, double, extended...*

Caràcter: Les variables d'aquest tipus poden contenir únicament un caràcter (`*', `a', `e', `~'...), s'expressa gràcies al *codi ASCII** ampliat.

Nom: *char*.

No té variacions, un caràcter ocupa sempre el mateix.

Lògics: Aquests tipus poden prendre únicament dos valors: vertader (*true)* o fals (*false*).

Nom: *boolean*.

Tampoc no té variacions, sempre ocupen 1 byte.

Definides pel programador: Aquelles que defineix l'usuari, és a dir, els dóna una extensió.

Subrang: Es defineixen a partir d'un tipus estàndard (excepte els reals), especificant dos valors constants d'aquest tipus, que seran els límits inferior i superior del conjunt de dades del subrang. Per exemple, podem

definir un subrang numèric (nombres enters) que vagi del 6 al 11. Els valors que formaran part del subrang seran 6, 7, 8, 9, 10 i 11.

En ser definit pel programador, ell és qui li posa el nom.

Enumerats: Aquests tipus són constituïts per un conjunt de valors referenciats per identificadors. Aquests valors formen una llista d'identificadors que el programador ha de definir. Per exemple un enumerat es pot dir Setmana i estar constituït pels valors dilluns, dimarts, dimecres, dijous, divendres, dissabte i dumenge.

- *Tipus de dades estructurades*

Cadena: És una seqüència de caràcters corresponents al codi ASCII, va escrita entre apòstrofs. També es considera una cadena quan no hi ha cap caràcter escrit, llavors se'n diu que és una cadena buida i es representa per dos apòstrofs seguits (``').

Nom: *String*

Conjunts: Es té, en programació, el mateix concepte de conjunt matemàtic i el tipus de dades conjunts. I les operacions que es realitzen amb ells també són les mateixes (intersecció, unió...).

Nom: *set*

Amb Pascal els elements del conjunt s'indiquen entre gafets i separats per comes ([3, 5, 7, 9]) i el màxim nombre d'elements que s'accepten en un conjunt és 256.

Arrays: És un conjunt finit i ordenat d'elements homogenis. És ordenat, perquè podem accedir a cada element de l'array de manera independent ja que podem fer referència a cada un d'ells utilitzant un *índex*. És homogeni perquè tots els elements són del mateix tipus de dades i és finit perquè té un nombre determinat de dades.

Existeixen dos tipus d'arrays: els unidimensionals (*vectors*) i els bidimensionals o multidimensionals (*matrius*).

- **Arrays: vectors i matrius**

Els *array* (conjunt finit, ordenat i homogeni de dades) són estructures de dades. Podem actuar (modificar, fer referència...) sobre un dels elements mitjançant *índexs*.

Hi ha dos tipus d'array: *vector* i *matriu.*

- ***Vectors.***

Són arrays unidimensionals, ja que per fer referència a qualsevol d'ells únicament ens cal un índex. Normalment com a índex s'utilitzen nombres enters (1, 2, 3, 4 ..).

Per dirigir−nos a un element d'un vector utilitzarem el nom de l'array i, entre gafets, l´índex que determina la posició de l'element en l'array. Per exemple, imaginem−nos que hem definit un vector amb el nom any i cada un dels seus elements és un mes d'aquest, si volem referir−nos al mes d'abril haurem d'escriure any[4] ja que l'abril és el quart més de l'any.

Alhora de definir un array sempre haurem de donar el nom, el *rang* dels seus índexs (extensió o nombre

d'elements que té) indicant sempre el primer i l'últim índex i el tipus de dades que conté. Per fer−ho s'utilitza la següent nomenclatura:

<nom> : array [i1..ip] de <tipus>

L'exemple anterior, en el llenguatge Pascal quedaria definit així:

any: *array*[1..12] *of* string;

Ja que l'any té dotze mesos, i tots els elements són del tipus string perquè volem que cada un contingui el nom del mes. Si volguéssim que cada un contingués el que hem gastat cada més, per exemple, en comptes de string hauria de ser integer.

Per fer−nos una idea mental del que és un vector ens podem imaginar de quina manera queda emmagatzemat en la memòria de l'ordinador:

Nom: any.

índex

any[1] any[2] ...

1	2	3	4	5	6	7	8	9	10	11	12

elements

FIGURA 9 *Esquema d'un vector en la memòria de l'ordinador.*

- ***Matrius***

Són aquell tipus d'array de més d'una dimensió i poden ser *bidimensional* (2 dimensions) o *multidimensionals* (tres o més dimensions).

Matrius bidimensionals: Són arrays (conjunt finit d'elements, ordenat i homogeni) i el que els diferencia dels vectors és que cada element es referencia mitjançant dos índexs. És el mateix concepte que es té en matemàtiques d'una matriu (un conjunt de M*N elements, tots del mateix tipus, cada un dels quals es referencia a través de dos subíndexs. El primer podrà valdre entre 1 i M i el segon entre 1 i N). Ens podem imaginar l'estructura d'una matriu d'aquest tipus de la següent manera:

Una matriu A de dimensió M*N. (M=4 i N=3)

A11	A12	A13	A14
A21	A22	A23	A24
A31	A32	A33	A34

FIGURA 10 *Estructura imaginària d'una matriu.*

En La figura anterior parlem d'estructura imaginària perquè l'ordinador, realment ho emmagatzema tot successivament (A11, A12, A13, A14, A21, A22, A23, A24, A31, etc,) com l'estructura d'un vector (figura 9).

L'ús de matrius és molt útil quan amb els vectors ens quedem limitats. Per exemple, imaginem que volem emmagatzemar a la memòria de l'ordinador les despeses que hem produït cada mes de l'any, però no només

d'un sol any sinó que ho volem saber dels últims cinc anys.

Podríem crear cinc vectors diferents cada un amb el nom d'un any (1995, 1996 ...). En aquest cas potser si que no seria gaire pesat de fer, però si volem fer el mateix amb una empresa, des que es va fundar fins l'actualitat Podríem crear 20, 30, 70 o més vectors amb els dotze mesos i això no és gaire útil. Per això és molt més senzill crear una matriu en la que M seria 70 i N seria des de l'any que es va fundar fins l'any actual, per exemple des de 1940 fins al 2000.

La definició d'una matriu es fa seguint la següent nomenclatura:

<nom> : *array* [1..M, 1..N] *de* <tipus>

L'exemple anterior (el de l'empresa), en Pascal, quedaria declarat de la següent manera:

Despeses: array[1..12, 1940..2000] *of* integer;

Matrius multidimensionals:

És una matriu de tres o més dimensions, encara que no és recomanable utilitzar−ne més de tres. Segons la dimensió de la matriu (3, 4, 5...) tindrem més o menys índexs, sempre coincideixen nombre d'índexs amb el de la dimensió de la matriu (la dimensió indica el nombre d'índexs i viceversa).

Imaginar−nos mentalment un array d'aquest tipus és molt complicat, això sí, en l'ordinador s'emmagatzema com l'anterior, consecutivament. I la seva declaració segueix la següent nomenclatura:

<nom> : *array* [1..m, 1..n, 1..p] *de* <tipus>

- **Constants i variables**
- Constants:

Una constant és un valor que no pot canviar durant l'execució del programa. El valor es defineix a l'inici de la compilació i ja no varia.

Les constants poden ser, principalment, de dos tipus: *literals* i *declarades*. Les literals són els valors que, el llenguatge, reconeix automàticament (1, 2, 5...), és a dir, que si escrivim el nombre 5 el llenguatge li associa el valor 5 i si escrivim b ho reconeix com el caràcter b.

Les declarades, són aquelles que s'identifiquen per un nom (identificador) i el valor que li assignem. Per exemple, el nombre ð (encara que està definit en la majoria dels llenguatges) . Per treballar amb aquest nombre en un programa, haurem de definir−lo com a constant (que li direm Pi) si ens volem estalviar haver d'escriure 3,14159... cada vegada que el vulguem utilitzar. D'aquesta manera només caldrà escriure *Pi* i el compilador ho reconeixerà com al valor real del nombre ðð

La nomenclatura que s'acostuma a seguir en declarar−les és:

<identificador> = valor; *(Pi = 3,14159;)*

Depenent dels llenguatges, algunes vegades s'ha de definir també el tipus de dada, però la majoria dels llenguatges prenen el tipus (string, integer...) segons el valor que li assignem. En el cas del nombre *pi* prenen el tipus *real* o alguna de les seves variacions.

- Variables:

Són aquells elements d'un programa als que li assignem un valor que pot canviar durant l'execució d'aquest. Totes les variables, com les constants no literals, han de ser declarades abans d'utilitzar−les. En la declaració si que s'ha de definir el tipus de dades que s'utilitzaran ja que, el compilador, en detectar−ne una, reserva un espai en la memòria, en funció del tipus, on s'emmagatzemarà el valor que li serà assignat.

Les variables, com les constants declarades, s'identifiquen amb un nom (identificador). Convé que aquest nom sigui significatiu, això fa el programa més llegible i comprensible per a nosaltres i també és recomanable utilitzar *comentaris* per aclarir com s'utilitza la variable. Una de les propietats de les variables és que els hi podem assignar qualsevol tipus de dades (cadenes, nombres, arrays...).

La nomenclatura que se segueix en declarar una variable, en Pascal, és la següent:

<variable (nom)> : tipus

Una característica important referent a les variables és l'operació *d'assignació*. Atorguem aquest nom a l'acció d'atribuir un valor a una variable, que pot ser una constant, una altra variable...

Imaginem que tenim dos variables en les que volem atribuir les mesures de dos segments (a i b) si les mesures de a i b fossin 4 i 6 centímetres respectivament, faríem l'assignació següent:

A 4

B 6

En Pascal, l'operador ( ) se simbolitza per : =.

Cal destacar que, quan assignem una cadena a una variable s'ha de fer entre apòstrofs.

Exemple: Dia := `Dimarts';

- **Expressions i operadors**

Una *expressió* és una combinació de variables, constants, signes d'operació, parèntesi i *noms especials* (Noms de *funcions estàndards*) normalment assignat a una variable. El resultat d'aquest conjunt d'elements esdevé un únic valor (una paraula, un nombre...).

Per exemple, sumar 3 a un nombre introduït per l'usuari, fer−li l'arrel quadrada i després multiplicar−lo per la suma d'uns altres dos és una expressió aritmètica. El pseudocodi de l'expressió seria el següent:

*Arrel(3 + A) * (B + C);*

Aquesta expressió està composta per les variables A, B i C (a les que l'usuari haurà d'assignar un valor), la constant literal 3, els signes d'operació + i *, uns parèntesis () que defineixen l'ordre de les operacions i el nom de la funció estàndard Arrel que calcula l'arrel quadrada del que hi ha dins el primer parèntesi. Tot aquest conjunt d'elements retorna únicament un valor (el resultat de fer les operacions). Aquest valor, normalment s'assigna a una variable, però també es pot mostrar directament a la pantalla o imprimir, per això diem que es pot assignar o no a una variable.

Totes les expressions tenen associat un tipus. Segons aquest poden ser *numèriques* o *lògiques.*

- *Expressions numèriques: operadors aritmètics*

Són Aquelles que utilitzen operadors aritmètics (+, *, /, ...) . Els elements utilitzats (variables i constants) també són numèrics, i el valor que retornen (que s'obté com a resultat) és també un nombre.

A part dels operadors senzills (+, *, −, /) els altres operadors que es poden utilitzar són per a realitzar l'operació *divisió entera* i l'operació *resta*.

Divisió entera: Ens retorna el quocient d'una divisió entera (en la que no es calculen decimals).

Nom: *div*

Exemple: 5 div 3 = 1

Operació resta: El valor que s'obté és la resta d'una divisió entera (el resultat és un nombre enter, no retorna decimals).

Nom: *mod*

Exemple: 5 mod 3 = 2

El nom que he escrit és el que els atorguen Pascal i *Object Pascal* un llenguatge molt semblant al pascal (utilitza els mateixos símbols i realitza les declaracions de la mateixa manera) però adaptat a la programació orientada a objectes.

Seguidament, podem observar una taula amb el tipus de nombres que s'han d'utilitzar en les expressions aritmètiques segons l'operació que s'utilitza i de quin tipus és el valor que es retorna.

Operador	***Operació***	***Valors***	***Resultat***
+	suma	enter/real	enter/real
−	resta	enter/real	enter/real
*****	multiplicació	enter/real	enter/real
/	divisió	enter/real	enter/real
div	divisió entera	enter	enter
mod	resta entera	enter	enter

FIGURA 11 *Tipus de valors utilitzats i resultats obtinguts en cada operació aritmètica.*

L'ordre de precedència (ordre en el que es realitzaran les operacions) es controla, com en matemàtiques, mitjançant parèntesis.

Exemple: (3 + 4) * 7 o també 3 + (4 * 7).

En el primer cas el resultat serà 49 i en l'altre 31. Per això indicar l'ordre de precedència és important, encara que ja està establert un ordre de precedència pels llenguatges, pràcticament igual al de les matemàtiques. D'aquesta manera, en la segona expressió de l'exemple anterior, no caldria utilitzar els parèntesis ja que l'ordre de precedència establert realitza abans la multiplicació que la suma.

S'ha de dir, però, que l'operador + també pot treballar amb valors de tipus cadena (string) i el resultat és un altre valor del mateix tipus.

Per exemple, si volem assignar el nom i el cognom d'una persona (Ramon Ibáñez) a dues variables i a una tercera variable assignar−li el nom i el cognom alhora, farem el següent:

Nom := `Ramon';

Cognom := `Ibáñez';

NomComplet := Nom + ` ` + Cognom;

El resultat obtingut en l'expressió assignada a la variable *NomComplet* és el nom i el cognom de la persona separats per un espai.

El tipus d'expressió d'aquest exemple també és aritmètica encara que en aquest cas no utilitzin nombres, l'operador utilitzat és la suma.

- *Expressions lògiques: operadors de relació i lògics.*

Són aquelles que tant sols poden retornar dos valors (vertader o fals). Els operadors que poden aparèixer en una expressió lògica són de dos tipus: *lògics* o *de relació*.

Mentre que en les expressions numèriques únicament es retornen i s'utilitzen valors numèrics, en les expressions lògiques els valors amb què es treballa per obtenir un resultat booleà no han de ser forçosament booleans. Això passa quan s'utilitzen operadors de relació.

Quan s'utilitzen *operadors de relació* es treballa sempre amb valors numèrics i el resultat obtingut és un valor booleà.

En canvi, quan els operadors són *lògics*, els valors amb els que es treballa han de ser sempre booleans i el resultat és un altre booleà.

Operadors de relació

Són aquells que relacionen dos parts o valors. Les relacions, en Pascal són:

Operadors Relació

< més gran que

> més petit que

= igual que

<> diferent de

" més gran o igual que

" més petit o igual que

FIGURA 13 *Tipus d'operadors de relació*

El tipus de relació ens obliga a treballar sempre amb valors numèrics.

Aquestes expressions ens permeten també treballar amb altres expressions, sempre que aquestes siguin numèriques.

La sintaxi d'aquest tipus d'expressions és sempre:

<Valor 1> operador <Valor2>

Exemple: 3*2 +2 > (4+2)/2 + 2

El resultat en aquest cas seria vertader (*true)* ja que 8 és més gran que 5.

Operadors lògics:

S'utilitzen quan volem avaluar una expressió amb més d'una condició al mateix temps (quan hem d'executar una ordre depenent de dos o més factors).

Per exemple, suposem que un programa ens calcula la divisió de dos nombres introduïts per l'usuari, però el numerador es transforma en la seva arrel quadrada. L'operació seria: *arrel(A) / B.* Per fer aquesta divisió sense cap problema, l'execució d'aquesta operació dependrà de dues condicions: que el denominador sigui diferent de zero i que A no sigui negatiu.

Els operadors que s'utilitzen en aquest tipus d'expressions són: i (*and*), o(*or*), no(*not*).

L'exemple anterior, en Pascal quedaria de la següent manera:

(B <> 0) *and* (a " 0)

Si els nombres introduïts per l'usuari compleixen aquestes dues condicions la divisió s'executarà.

- ESTRUCTURA GENERAL D'UN PROGRAMA
- **Parts d'un programa**

Un programa és un conjunt d'instruccions que, en ser executades ordenadament, resolen un problema.

Cada programa té quatre parts principals (es poden veure ben diferenciades en la figura 4):

- Declaracions: Consisteix en declarar les variables, constants... que s'utilitzaran al llarg del programa
- Entrada de dades: Consisteix a demanar a l'usuari l'entrada de dades necessàries per l'execució del programa. Es fa a través d'instruccions de *lectura*. Que consisteixen a ordenar al programa que reconegui el que l'usuari ha escrit a la pantalla i assignar−ho a una variable.

(En la figura 4 l'entrada de dades seria la lectura dels dos nombres enters i assignar−los a les variables A i B).

- Accions de l'algorisme: Part en la que es resol el problema utilitzant les dades d'entrada a partir de la traducció del pseudocodi (o altres representacions) de l'algorisme.
- Sortida: Consisteix en mostrar en un dispositiu de *sortida* els resultats de les accions realitzades anteriorment. Aquestes accions s'anomenen *accions d'escriptura.*
- **Elements bàsics d'un programa**

A part de les variables, constants, comentaris i identificadors hi ha uns altres elements fonamentals en la creació d'un programa: els *caràcters especials* i les *instruccions*.

Els *caràcters especials* serveixen com a separadors de *sentències* (assignació, declaració, etc). Per exemple el caràcter ; s'escriu sempre, després de fer una assignació, de declarar una variable, una constant, un *procediment, etc*.

Les *instruccions* o sentències són cada un dels passos d'un algorisme, però executat per l'ordinador. Poden ser d'inici i final del programa, d'entrada i sortida, assignacions etc,

Totes aquestes instruccions formen tres grans grups, quan les classifiquem, segons la seva execució. Els tres tipus d'instruccions principals (segons l'execució d'aquestes) són les *seqüencials, selectives* i *repetitives*.

- Seqüencials: Aquell grup d'instruccions que s'executen o permeten l'execució directa d'una rera l'altra.
- Selectives. Aquelles instruccions que imposen una condició per executar o no les instruccions que venen després o unes altres.
- Repetitives: Aquelles instruccions que repeteixen un nombre d'instruccions un nombre finit de vegades.

- **Escriptura d'un programa**

L'escriptura d'un programa té dues parts: L'*encapçalament* i el *cos*. La primera conté el nom de l'algorisme. La segona part, el cos, està format per dues parts més.

La primera part del cos és la declaració de *llibreries** constants i variables i la segona està destinada al conjunt d'instruccions del programa.

En la zona de les instruccions, per què el programa quedi més llegible s'han d'utilitzar, com hem dit abans, identificadors significatius i és molt recomanable l'ús de comentaris (normalment s'escriuen entre claus).

Si ens fixem en l'exemple de la figura 14, les parts del programa i de la seva escriptura es veuen força clares (gràcies a l'ús de comentaris).

Un comerciant vol un programa que: ingressi el cost del producte comprat, que li incrementi un 30% de beneficis i que li incrementi també un 16% d'IVA. També vol que li calculi el preu al que vendrà el producte.

Pseudocodi

INICI

Definir constants (IVA i benefici)

Llegir el cost del producte

Incrementar al cost un 30%

Calcular el valor de l'IVA

Calcular el preu

Escriure a la pantalla:

- *El cost*
- *L'impost*
- *El preu de venda*

FINAL

Programa escrit en Pascal

```
{* programa de gestió *}

uses

crt, dos; {* declaracions de llibreries *}

const

ben = 30;

iva = 16; {* declaracions de constants *}

var

cost: real;

cost2, impost, preu: real; {* declaració de variables *}

begin {* inici del programa *}

clrscr; {* neteja de la pantalla *}

write(`Ingressi el cost del producte:'); {* sortida de missatge *}

readln (cost); {* ingrés de dades *}

cost2 := cost * (1 + ben / 100); {* càlcul del benefici *}

impost := cost2 * (iva / 100); {* càlcul de l'impost * }

preu := cost2 + impost; {* càlcul del preu final *}

writeln(`Cost inicial: `, cost); {* escriptura dels resultats *}

writeln(`Impost: `, impost);

writeln(`Preu final:', preu);

end. {* fi del programa *}
```

FIGURA 14 *Escriptura d'un programa de gestió en Pascal*

En els punts *4.3 (Programació modular) i 4.4 (Funcions i procediments) es fa una explicació d'aquests conceptes.*

És un programa que es troba dins d'un altre (punt 4.3)

Són pròpies de les estructures repetitives de programació (punt4.2)

Són aquells llenguatges en els que totes les dades utilitzades han de tenir els seus propis tipus declarats explícitament.

Son diferents noms que se li donen a un mateix tipus de dades, de manera que cada un accepta un nombre diferent de xifres, caracters... i en funció d'això ocuparà més memòria o menys.

Mirar la operació d'assignació de la pàg 23.

Aquest tipus d'estructura de dades surt més explicat al punt següent (2.4)

Són textos aclaridors per qui llegeix el còdi del programa. El compilador prescindeix d'ells.

Aquells dispositius de l'ordinador que permeten la sortida de dades (impressora, monitor ...).

Estàn molt rel·lacionades amb els tipus d'estructures de programació (punt 4.2)

5

La programació d'ordniadors Artur Sales

La programació d'ordinadors Artur Sales

29

CPU

IMPRESSORA

MÒDEM

TECLAT

ESCÀNNER

MOUSE

MONITOR

N => 0

Llenguatge d'alt nivell

Compilador o intèrpret

Llenguatge màquina

FIGURA 2 *Una de les primeres màquines programada mitjançant la modificació del seu hardware.*

LECTURA DEL NOMBRE (N)

ESCRIVIM EL VALOR EN LA PANTALLA

CÀLCUL DE L'ARREL

INICI

Volem calcular l'arrel quadrada d'un nombre.

si

no

ABORTEM

ESCRIVIM L'ERROR EN LA PANTALLA

FINAL

FIGURA 5 *Diagrama de fluxe.*

INICI

LLEGIR EL NOMBRE (N)

N = > 0

SI NO

CÀLCUL DE

L'ARREL

ESCRIVIM

L'ERROR A LA

ESCRIVIM EL PANTALLA __VALOR A LA

PANTALLA

FINAL