Passat, present i futur de la programació

Tothom sap que l'acció humana sobre les màquines és de dos tipus: *a*) en primer lloc, cal construir-les, dissenyar i fabricar suports tècnics que facin realitat els ordinadors; i *b*) cal fer anar els ordinadors, a partir del seu disseny tècnic concret, a partir d'un llenguatge de comunicació, el "programa". Aquest treball ens parla de l'evolució de la programació i ens descriu els principals elements i les possibilitats actuals de la relació "comunicativa" que podem tenir actualment amb els ordinadors.

Definicions i opinions

Si busquem el mot *programa* en un bon diccionari, trobem, si fa no fa, una definició tal com "enunciat del que es vol o es pensa fer". I els mots *programació* i *programador,* com a derivats, estaran definits com a "acció de..." i "persona que...". Són uns mots, però, que en l'època actual s'usen en un gran nombre de contextos diferents i amb significants que, si bé deriven de la definició original, s'apliquen a conceptes força diferents. Fins i tot, en camps estretament vinculats a la informàtica, com pot ser la investigació operativa, trobem tècniques i mètodes anomenats *programació matemàtica, programació dinàmica,* etc... Cal, doncs, delimitar el significat del mot *programació* en informàtica, per a la qual cosa introduïm tot seguit algunes definicions.

Un computador (o, com veurem, qualsevol giny programable, com, per exemple, un robot) pot fer diferents tasques; i pot fer tasques diferents perquè disposa d'un repertori d'operacions elementals que sap fer a alta velocitat i/o amb gran precisió. Però, en realitat, només podrà fer aquelles tasques que poden ser especificades en termes de les operacions elementals que sap executar. Si volem, doncs, fer una tasca determinada amb un computador, haurem de definir quines operacions elementals, i amb quin ordre, cal fer; en termes més amplis, haurem de descriure com fer aquesta tasca. D'aquesta descripció en diuen *algorisme.* Naturalment, un algorisme necessita una notació (textual, gràfica,...) per ser descrit, de la qual en direm *notació algorísmica,* amb el benentès que una notació no té perquè servir per descriure tota classe d'algorismes: segons la classe d'algorisme (la classe de problema), una notació pot ser millor que una altra o, senzillament, no servir. Com més formal és aquesta notació, més ens permet el tractament de l'algorisme com a objecte matemàtic (per provar, per exemple, la seva correctesa).

L'algorisme és una descripció estàtica d'un concepte dinàmic: quan executem les tasques descrites per l'algorisme, estem executant un procés. De l'agent capaç d'executar processos se'n diu, en termes generals, *processador.* Un processador pot ser una persona o un giny mecànic o electrònic. Quan es tracta d'un computador, l'algorisme que descriu el procés ha d'estar escrit en una notació intel·ligible per a la màquina. D'aquestes notacions en diem *llenguatges de programació.* De l'algorisme escrit en un llenguatge de programació en diem *programa* i, en conseqüència, de l'activitat de construir programes en diem *programar.* No podem, però, donar la definició de programació com a "acció de programar", encara que també tingui aquest sentit. L'evolució de la informàtica ha portat tècniques i mètodes per fer programes, a més dels llenguatges. I tant els mètodes com els llenguatges requereixen bases formals on recolzar-se. Tot això configura el significat de *programació* en el nostre context: disciplina científica que estudia les bases teòriques necessàries sobre les quals fonamentar mètodes i llenguatges de suport a la construcció de programes. Cal posar èmfasi en aquesta idea, inseparable de qualsevol concepció moderna de la informàtica. I si cal posar-hi èmfasi és perquè es manté encara fortament una concepció estrictament utilitària de la programació, una concepció per a la qual *programació* és *únicament* l'activitat de fer programes, i que lògicament està molt més estesa (lògicament, ja que la programació-activitat és coneguda en molts àmbits de la societat; en canvi, la programació-disciplina científica, tot i donar suport a la primera, és coneguda només en els petits cenacles de la universitat i d'algun centre de recerca). D'aquesta concepció neix una confusió que està esdevenint crònica i, fins i tot, perillosa: la confusió entre programació i llenguatges, o precisant més, la no distinció entre la fase de construcció de l'algorisme i la de la seva codificació en una notació adequada. I, com a conseqüències, dues opinions (si més no) molt esteses, i, sota l'òptica científica,[1] greument errònies:

a) La capacitat o potència de la programació es mesura per les possibilitats dels llenguatges. Un llenguatge és més "bo" com "més coses" permet fer.

b) Ensenyar a programar consisteix a explicar l'ús del llenguatge que serà usat.

Les opinions del món científic són, en referència a les anteriors, les següents:

a') La potència de la programació està en la capacitat per dissenyar algorismes adequats als problemes que es pretén resoldre. La bondat d'un llenguatge es mesura per la seva adequació a una classe de problemes, la seva fiabilitat i l'aprofitament que fa dels cursos del *hardware,*[2] entre d'altres aspectes.

b') Ensenyar a programar consisteix a transmetre una forma sistemàtica i rigorosa de dissenyar algorismes, maximitzant la fiabilitat i l'eficiència del producte obtingut. L'aprenentatge dels llenguatges, vehicles per descriure programes, s'ha de fer després (o en paral·lel) a l'aprenentatge de la programació.

Com ja hem dit, aquestes darreres opinions són minoritàries, cosa que tindria una fàcil explicació si tan sols fossin qualificables, pejorativament, d'academicistes. Però mantenir les opinions contràries significa un considerable increment de costos a les indústries (s'estima que, en una instal·lació gran, els programadors ocupen el 80% de la seva vida professional a corregir errors i fer

per Pere Botella

Pere Botella (Barcelona, 1949) és enginyer industrial i doctor en informàtica. Va treballar com a suport informàtic en treballs de planificació urbanística. A la Universitat Politècnica, ha treballat en el seu centre de càlcul, i des del 1977 és professor del departament de programació de la Facultat d'Informàtica, on, tant en docència com en recerca, es dedica als camps de la teoria, la metodologia i els llenguatges de programació.

Nota: El present article pot ser complementat per l'entrevista amb E.W. Dijkstra, realitzada per Albert Llamosí i Ferran Orgàs, en ocasió de la visita del professor a la Conveció Informàtica Llatina d'aquest any. Aquesta entrevista, que us reconanem, serà publicada properament en aquestes pàgines.

modificacions en els programes), encara que no sigui evident, a primera vista, que un canvi d'enfocament ajudi a fer decréixer els costos (el problema no és sols de costos econòmics: pensem en les aplicacions crítiques dels computadors, com el control de plantes energètiques, ginys per a la guerra, etc...). Deixem, però, aquest tema per reprendre'l al final de l'article.

Dels primers llenguatges a la renovació de conceptes

Tant els primers ordinadors com la major part dels d'avui dia (incloent-hi els populars microordinadors) responen a una mateixa arquitectura, que rep el nom d'"Arquitectura Von-Neumann" en record d'un dels seus principals dissenyadors: és el ja popular esquema[3] que reproduïm a continuació

En aquestes màquines, el programa, emmagatzemat a la memòria, està format per una sèrie d'instruccions, molt simples, del tipus

Codi operació Adreça 1 Adreça 2

i hi existeix un comptador d'instruccions indicant quina instrucció s'executa, i un decodificador que interpreta el seu significat i activa els circuits necessaris per executar-la. Aquestes instruccions, expressades en forma numèrica (binària, octal, hexadecimal...), diuen que cal fer-les (codi d'operació) amb els (o l') operands continguts a les adreces 1 i 2. Per exemple, si el codi 10 és "guardar", 11 "sumar" i 12 "multiplicar", podríem tenir el següent tros de programa (amb el benentès que es tracta d'una màquina imaginària):

```
10  100  200   —copia el contingut de
                 100 a 200
12  120  200   —, el multiplica pel con-
                 tingut de 120
11  130  200   —, li suma el de 130
10  200  100   —i el resultat es diposita a
                 100
```

Les instruccions s'executen seqüencialment, però si cal trencar la seqüència es disposa de salts incondicionals ("la següent instrucció serà la nº n") i condicionals ("si la condició c és certa executarem la instrucció n, i si no, la següent").
Com és evident, programar directament amb aquest llenguatge numèric (anomenat *llenguatge màquina*) es va fer ràpidament antipàtic, sobretot a mesura que els programes eren més complexos. Així varen aparèixer els *llenguatges assembladors,* que substituïen els codis d'operació per mnemònics i les adreces de memòria per identificadors. D'aquesta manera el programa anterior agafaria una forma tal com:

```
GUARDA      a, R
MULTIPLICA  b, R
SUMA        c, R
GUARDA      R, a
```

Dels traductors a llenguatge màquina se'n diu, també, *assembladors.* Amb ells apareix el primer pas vers l'abstracció: l'adreça concreta se substitueix per un símbol, la qual cosa permet que un mateix programa serveixi per a diferents adreces. Ràpidament va sorgir la idea de formar grups d'instruccions que sempre s'executaven en el mateix ordre i donar-hi un sol nom: per exemple, la lectura d'un caràcter des d'un dispositiu extern podia prendre la forma:

LLEGIR PANTALLA, a

i un cop traduït, generar diverses instruccions. D'aquestes instruccions se'n diu *macros,* i dels llenguatges que les contenen, *macroassembladors.*
Malgrat tot, la construcció de programes encara es feia força pesada i, així, varen començar a aparèixer els primers intents de llenguatges *d'alt nivell* (per contraposició als assembladors, dits de *baix nivell*). El primer a consolidar-se va ser el FORTRAN, aparegut el 1957 (i criticat llavors com a producte acadèmic, poc útil per als problems "reals"). Amb ell es va fer un nou pas vers l'abstracció, en poder escriure expressions aritmètiques en una sola instrucció i en introduir subprogrames que podien ser invocats des d'un altre programa. Per exemple, la seqüència d'instruccions anterior s'escriuria en FORTRAN com a

A = A * B + C

i la coneguda fórmula $A = \sqrt{b^2 - 4\ ac}$, com a

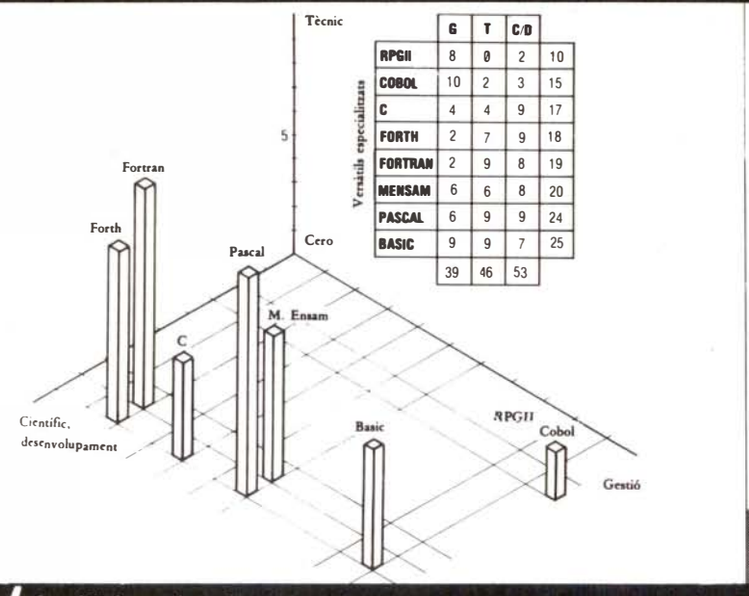
DISC = SQRT (B**2 - 4.0 *A * C)

on SQRT és el nom d'un subprograma que calcula l'arrel quadrada, i que porta incorporat el llenguatge. Si, per exem-

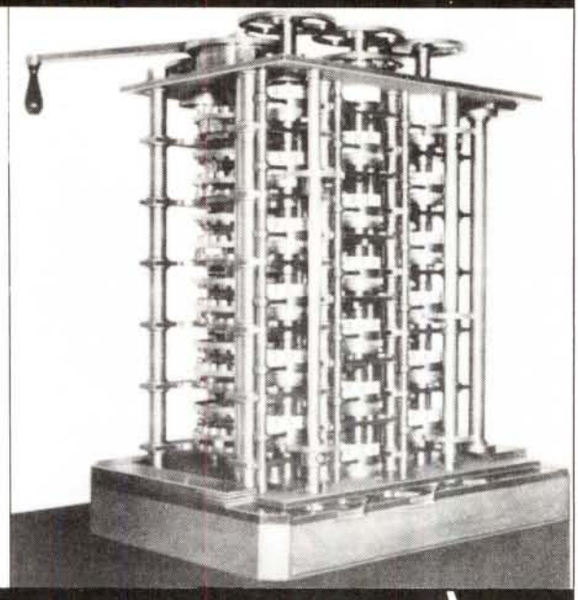
1. Em refereixo a la ciència informàtica. Dissortadament, científics d'altres disciplines també sostenen aquestes opinions.

2. Conjunt d'elements físics que componen un computador; el mot *software* fa referència al conjunt dels programes.

3. En un recent número de "Perspectiva Escolar" (Rosa Sensat, juny del 1982) vaig fer una descripció d'aquest esquema, en forma molt simple, com a proposta per ser usada en els darrers cursos d'EGB.



	G	T	C/D	
RPGII	8	0	2	10
COBOL	10	2	3	15
C	4	4	9	17
FORTH	2	7	9	18
FORTRAN	2	9	8	19
MENSAM	6	6	8	20
PASCAL	6	9	9	24
BASIC	9	9	7	25
	39	46	53	



ple, necessitéssim la funció "valor absolut" (i el llenguatge no la tingués incorporada), escriuríem:

```
      REAL FUNCTION ABS (x)
      IF (x.LT.O.) GO TO 1
      ABS = X
      RETURN
1     ABS = -X
      RETURN
      END
```

Com podem veure en l'exemple, el control de la seqüència es manté en el nivell dels assembladors, amb salts condicionals i incondicionals. El FORTRAN, que va ser dissenyat per tractar problemes numèrics, va ser adoptat com a llenguatge estàndard pel departament de Defensa dels EUA (fet que, com veurem més endavant, té una molt particular importància).

Aquest mateix departament, davant d'un gran volum de tasques de gestió administrativa i de la inadequació del FORTRAN per a aquestes, va dissenyar el COBOL (1960), fita important perquè significa un primer intent de llenguatge pensat per resoldre una classe de problemes (els administratius) fent abstracció de la forma del llenguatge màquina. Dues sentències exemple ens poden ajudar a comprendre el que diem:

```
MULTIPLY UNITARI BY
QUANTITAT GIVING PREU.
o bé
MOVE FACTURA TO SORTIDA.
```

L'execució de programes escrits en COBOL, FORTRAN o en un altre llenguatge d'alt nivell requereix l'existència d'un programa traductor, dit compilador, que llegeix el text del programa, en detecta els errors sintàctics i, si no n'ha trobat, el tradueix a llenguatge màquina.

Els compiladors d'aquests primers llenguatges es complicaven seriosament en haver de tractar sintaxis carregades d'ambigüitats, excepcions i casos especials. La publicació del report sobre el llenguatge Algol 60 va significar la introducció d'un metallenguatge, el BNF, que permetia descriure la gramàtica de l'Algol en forma precisa i no ambigua, amb el que això significava per a la construcció de compiladors. L'Algol, que va ser pensat inicialment per descriure algorismes a les publicacions (donada la deficient legibilitat del FORTRAN), va consolidar l'abstracció funcional i l'abstracció en el control, aquesta darrera amb sentències del tipus *if-then-else, for,* l'estructura de blocs, etc..., deixant el model imposat per la memòria i constituint-se en el precursor de gairebé tots els llenguatges moderns de tipus imperatiu. D'altra banda, l'Algol va ser un intent de resposta europea al domini nord-americà en aquest camp. L'obsessió, però, de l'època (cap al 1965) era la d'arribar al llenguatge de propòsit general: que servís per a tot i que tingués prevista qualsevol situació, qualsevol tipus d'objecte. Dues en són les mostres més significatives: el PL/I i l'Algol 68.

El PL/I, producte d'una multinacional, va ser un intent de reunir en un sol llenguatge les característiques dels Cobol, Fortran i Algol, més els avenços més recents (nous tipus de fitxers, possibilitat de sincronització, etc.). Ràpidament es va constatar la dificultat que tindria el programador normal a dominar un llenguatge tan complex, i, en ser publicat, ja era un subconjunt de la seva definició. Per la seva banda, l'Algol 68, més ben definit però sense el suport de cap empresa, patia del mateix mal: un excés de complexitat el deixava al marge de l'abast del programador d'aplicacions. Però, tant PL/I com Algol, 68 varen ser bandejats per una onada que era independent de l'anàlisi estricta del llenguatge, i que veurem a l'apartat següent. Menció a part es mereixen dos tipus (molt *grosso modo*) de llenguatges:

a) En oferir les màquines la possibilitat de treball **interactiu** varen aparèixer llenguatges que aprofitaven aquest fet. El BASIC va ser creat per ensenyar a programar a nens de 10-12 anys i, en la seva versió original, era una mena de FORTRAN en miniatura. El programa traductor era un *intèrpret,* en comptes d'un compilador: residia sempre a memòria i permetia fer correccions i executar immediatament, tornar a corregir, etc... Una altra experiència, molt diferent, però també interactiva, va ser la del llenguatge APL, de Ken Iverson, format per una gran quantitat d'operadors (funcions), que converteixen el terminal d'un computador en una supercalculadora de butxaca, amb quantitat de "tecles" i amb la possibilitat de combinar-les, molt adequat per a programes de "teclejar-i-llençar".

b) En obrir-se nous camps d'ús (intel·ligència artificial, tractament de textos, etc.) varen dissenyar-se llenguatges *ad-hoc.* Especial importància té el LISP, de Mc Carthy, en donar origen a la saga de llenguatges funcionals o aplicatius, en contraposició als imperatius. Un programa LPS és una funció (en el pur sentit matemàtic) construïda combinant funcions elementals. En general, els programes funcionals descriuen la semàntica del que es vol fer mitjançant funcions (els programes imperatius descriuen quines accions fer, i en quin ordre, per obtenir el resultat). Citem aquí també el SNOBOL, per a tractament de textos, i el LOGO, per a ensenyament, amb un enfocament diferent dels llenguatges imperatius.

La programació esdevé disciplina científica

Cap a les darreries dels anys seixanta, inicis dels setanta, un grup de professors, majoritàriament europeus, entre els quals podem citar l'holandès Dijkstra, el norueg Dahl, els anglesos Hoare i Randell, el suís Wirth i el danès Naur, varen formar un grup de treball en el si d'una federació internacional d'informàtics (la IFIP) sobre el nou tema *programming methodology*. Davant de les cada cop més complexes màquines i aplicacions de la informàtica, la programació i les seves eines es feien insuficients, i disposar de llenguatges cada cop més potents i complexos no n'era solució: calia estudiar els fonaments teòrics de la construcció de programes i, com a conseqüència, dissenyar mètodes ben fonamentals que permetessin abordar problemes complexos maximitzant la fiabilitat del programa obtingut. De fet, fins aleshores, el programador disposava d'eines (llenguatges i traductors) i tècniques (gairebé completament recopilades en l'obra *The art of computer programming,* Knuth, matemàtic nord-americà), però li mancaven mètodes que li permetessin treballar amb sistemàtica i rigor.
Les propostes potser més remarcables de l'esmentat grup eren:
a) Ús de l'**abstracció** com a eina bàsica de disseny. Un programa s'ha de dissenyar partint d'un esquema abstracte, i, per refinaments successius, detallar totes les seves parts fins a obtenir la solució definitiva.
Per exemple, l'acció "canviar-una-roda" es refinaria a:

afluixar-rosques
elevar-cotxe
treure-rosques
treure - roda
posar - nova - roda
etc.

i "afluixar - rosques", per exemple, es refinaria per *repetir* 6 *vegades* "afluixar - una - rosca".
b) **Composició** de programes mitjançant tres estructures bàsiques
–seqüència acció$_1$; acció$_2$; —; acció$_n$
–selecció *si* condició *llavors* acció$_1$ *si no* acció$_2$
–iteració *mentre* condició *fer* acció
, corresponents a formes més pròximes al llenguatge natural.
c) Disposar d'eines **sintàctiques** per crear tipus de dades per modelitzar els objectes del món real, en comptes de representar-los en termes d'un repertori finit de tipus (encara que fos molt extens, com en PL/I).
d) Possibilitat de tractar els programes com a objectes **formals,** disposant d'eines per verificar la seva correctesa.
Aquestes propostes estan contingudes, entre d'altres treballs, en el llibre *Structured programming,* de Dahl, Dijkstra i Hoare, que, a més de donar nom a aquest moviment renovador, va proclamar l'obsolescència, si més no en l'àmbit acadèmic, dels llenguatges imperatius descrits en l'apartat anterior (incloses les propostes més recents, com PL/I o Algol 68, tot i el greuge de posar-los tots dos en el mateix sac).
El primer fill (i fill preferit) d'aquest moviment va ser el llenguatge Pascal (Wirth 1971), derivat de l'Algol 60, però que va canviar de nom per no confondre's amb l'Algol 68. En recollir les quatre propostes esmentades més amunt, va convertir-se ràpidament en l'esperanto de l'ensenyament universitari (de fet, va ser dissenyat per a l'ensenyament). Com a exemple de la seva legibilitat, vegem el clàssic algorisme d'Euclides per trobar el màxim comú divisor de *x* i *y*, escrit en Pascal:

```
var x, y, a, b: integer;
read (x, y);
a: = x; b: = y;
while a < >b do
  if a > b then a: = a – b
           else b: = b – a

write ('El MCD és', a)
(el símbol < > és "diferent de")
```

Podem comparar aquest programa amb la seva versió BASIC, com a mostra de la diferent expressió quan es disposa (Pascal) o no (Basic) d'abstracció en el control.

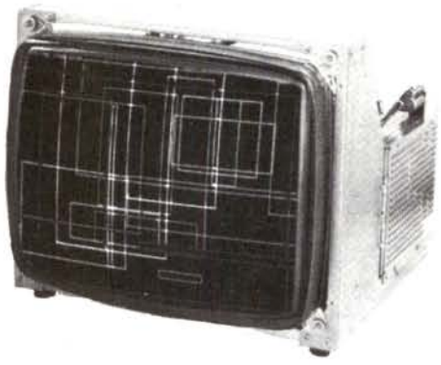
```
10 INPUNT x
20 INPUNT y
30 LET    A = X
40 LET    B = y
50 IF     (A EQ B) THEN 90
60 IF     (A GT B) LET A = A - B
70 IF     (A LT B) LET B = B - A
80 GO TO 50
90 PRINT  'EL M.C.D. ÉS', A
100 END
```

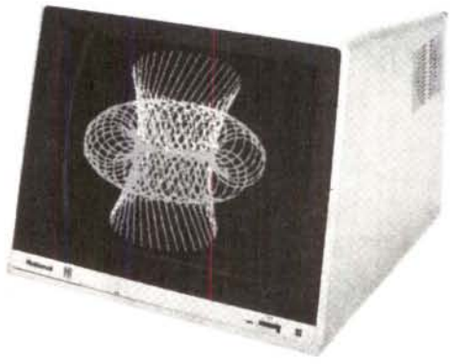
Però Pascal va ser un inici, no un final. La recerca en programació, durant tota la dècada dels setanta, ha estat molt activa i rica en aportacions. Podem citar, com a més remarcables:

a) **Recerques metodològiques.** Partint dels principis de l'abstracció i del disseny descendent (dels aspectes generals als particulars), s'han definit molts mètodes, adaptats a diferents classes de problemes.[4] S'ha treballat també en diferents intents d'obtenir un programa partint d'unes especificacions formals (derivació formal, transformació i síntesi de programes).

b) **Verificació de programes.** S'ha tractat d'aplicar diferents llenguatges formals (lògica de primer ordre, lògica temporal, àlgebres heterogènies) per provar la correctesa dels programes, abans de ser passats per màquina. Si bé

4. Els pensats per a gestió, com el definit per Jackson, són els que més s'han popularitzat en el nostre país.





les aplicacions pràctiques són, encara, petites, la *semàntica de llenguatges* ha rebut un important impuls.

c) **Llenguatges imperatius.** En la línia encetada per Pascal, però tractant d'incorporar nous conceptes sorgits de la recerca, varen aparèixer Euclid, Clu, Alphard, Gypsy, Mesa i molts d'altres. Del darrer en aquesta línia, el llenguatge ADA, en parlem més endavant.

d) **Llenguatges i programació funcional.** La línia iniciada pel LISP rep un nou impuls amb la proposta de Backus (dissenyador del FORTRAN) d'un estil de programació, FP (Functional Programming), que vol ser l'alternativa a l'estil imperatiu, forçat, segons Backus, per l'arquitectura Von Neumann. Anteriors o posteriors a aquesta proposta trobem els NPL, POP, SASL, KRC, HOPE, etc.

e) **Paral·lelisme.** Cada dia són més freqüents les aplicacions en les quals els programes no responen a un model seqüencial, sinó que són constituïts per un conjunt de processos executats en paral·lel i que treballen concurrentment per assolir l'objectiu desitjat. El model paral·lel va basar-se inicialment en processos independents que compartien recursos (situats a la memòria comuna) i se sincronitzaven per tal d'accedir-hi. Actualment, amb la tendència cap a arquitectures amb xarxes de microprocessadors, el model consisteix en processos independents que se "citen" per sincronitzar-se i intercanviar informació. S'han dissenyat llenguatges, en aquesta àrea, com el Concurrent Pascal, Modula o el mateix Ada.

f) **L'abstracció de dades.** El concepte de *tipus abstracte de dades* és, potser, l'aportació més rellevant dels darrers anys, i ha impregnat la recerca en totes, sense excepció, les àrees anteriors (metodologia, verificació, llenguatges imperatius i funcionals, paral·lelisme). Un tipus abstracte de dades defineix una classe de valors per a les operacions que s'hi poden fer: això es tradueix, en els llenguatges, a disposar de mecanismes sintàctics (dits d'*encapsulat*) que permetin definir conjuntament tipus de dades i funcions (o procediments) de manera que, des d'una altra part del programa, sols s'accedeix a les dades via aquestes funcions. Als avantatges pràctics (disseny descendent d'objectes i accions, modularitat, fiabilitat), s'hi afegeix l'elegància de la teoria algebraica que s'hi troba subjacent.

Finalment cal parlar del llenguatge Ada. El ja citat departament de Defensa dels EUA (el client més important de les multinacionals) disposava del Fortran i el Cobol com a llenguatges estàndards per a càlcul numèric i gestió, respectivament, però estava mancat d'un estàndard per a les aplicacions en temps real, control, xarxes, etc...

Després de convocar un concurs públic, el llenguatge guanyador va ser proclamat nou estàndard i anomenat Ada (en honor a Ada Augusta Byron, filla del poeta Lord Byron, i programadora, el segle passat, de la màquina analítica de Charles Babbage, precursora dels actuals ordinadors). El llenguatge Ada, dissenyat per un equip europeu, ha aconseguit encetar una viva polèmica: des dels qui el consideren modern (en adaptar les darreres aportacions de la recerca), potent i modèlic pel seu disseny; els qui en són detractors en veure-hi un recaure en els defectes imputats als PL/I o Algol 68, agreujat pel fet de ser dissenyat per a aplicacions crítiques com ho són les militars; i fins els pragmàtics, que opinen que, si bé no és l'òptim, més val adaptar-s'hi vist que el procés sembla imparable. Aquesta discussió, però, es manté, de moment, dins dels cercles especialitzats: la major part del món professional hi resta al marge.

Situació actual i tendències previsibles

Estem vivint un moment en el qual la informàtica ha deixat de ser una alquímia coneguda només per uns quants professionals que hi havien arribat per mitjà d'uns ritus iniciàtics i que parlaven un llenguatge obscur i críptic (encara que aquesta actitud encara ara sigui mantinguda per alguns; res més oposat, no sols a la ciència, sinó al rigor professional). Avui dia la informàtica és un fet quotidià que es manifesta en múltiples formes. Això ens fa molt difícil d'analitzar la situació actual de la programació d'una forma global, única; així doncs, passem a veure, separadament, alguns aspectes:

–*L'allunyament entre la programació com a disciplina científica, i la programació com a pràctica.* Al primer apartat he parlat de dues opinions, dues òptiques sobre la programació, i estem davant d'una conseqüència d'aquesta confrontació. Els llenguatges més utilitzats actualment a tot el món (llevat del cas del micros) són el FORTRAN (del 1957) i el COBOL (del 1960). El Pascal (del 1971), amb tot l'enrenou que l'envolta, ha començat a fer forat amb deu anys de retard... quan comença a ser, conceptualment, obsolet (l'obsolescència científica dels Fortran i Cobol és absoluta). Tot el que s'ha descrit en l'apartat 3 d'aquest treball, ara per ara, no ha fet més que tímids tocs en el món professional. Sembla paradoxal que, mentre en el món del *hadware* els nous avenços tracten de ser ràpidament assimilats per la indústria, en el món del *software* qualsevol novetat sigui vista amb recel, si no amb clara hostilitat. I si en els països productors de *software* aquests dos mons no tenen altre remei que anar, de tant en tant, de bracet, en els països consumidors (colonitzats?)

Material de lectura

O.J. Dahl, E.W. Dijkstra, C.A.R. Hoare: *Structured Programming.* Londres Academic Press, 1979.

D. Gries (Ed.): Programming Methodology (a collection of articles by members of IFIP WG 2.3), New York, Springer-Verlag, 1978.

C.A.R. Hoare: *The Emperor's old clothes,* "Communications of the A.C.M., "Vol. 29 n.º 2, Febrer, 1981.

T.W. Pratt: *Programming languages: design and implementation,* Prentice-Hall, 1982.

N. Wirth: *Algoritmos + Estructuras de datos = Programas, Madrid, Ed. del Castillo, 1980.*

Nota: Es recomana especialment la lectura dels articles de Hoare (citat) i de Dijkstra, The humble programmer, continguts en el recull d'articles de Gries.

com el nostre, és totalment inútil esperar que hi vagin.

Hi ha, però, diverses raons que, en part, expliquen aquest fet. Raons econòmiques (el cost de reprogramar tots els programes en explotació, per exemple) i raons socials (canvi de costums personals, prevenció de l'usuari que no vol que l'eina absorbeixi el seu temps, etc...). L'anàlisi d'aquest fet, però, requereix un marc diferent al d'aquest escrit.

Què passarà amb l'Ada?. Hi ha qui diu que res. Hi ha qui diu que, d'aquí a quatre dies, tots hi programarem. De fet, els dos llenguatges més usats són els dos estàndard de les forces armades dels EUA i Ada ve a ser el tercer estàndard. La situació d'ara, però, és molt diferent: Fortran i Cobol tenien el terreny pla per fer-se els guanyadors; Ada té, en canvi, competència (sobretot, per les raons de l'anterior paràgraf). En qualsevol cas, una de les tres armades dels EUA ja ha decidit reprogramar tot el que sigui escrit en Fortran i Cobol, o en qualsevol altre llenguatge. Tot en Ada. I les grans constructores no en són pas refractàries.

–*L'ordinador personal,* el microordinador, ha aprofitat, primer un i després l'altre, dos dels llenguatges més populars dissenyats per a ensenyament: el Basic i el Pascal. I els ha portat a una curiosa popularitat plena de marcianets, laberints, E.T.'s, cançonetes i, també, aprenentatge, comptabilitat domèstica i altres utilitats. El futur ens dirà si això és bo, dolent o innocu. De moment, per als comerciants, els va bé.

–*El programador tendeix a desaparèixer,* si més no, tal com l'entenem avui dia. Generadors de programes, com els RPG, o els "menús" de les actuals màquines de gestió, són precursors d'aquesta desaparició. L'ús de l'ordinador intenta fer-lo transparent a l'usuari, de manera que no s'hagi de ser especialista per servir-se'n. D'altra banda, amb la imparable irrupció dels micros a les escoles, tothom serà programador *amateur.* Així que, per a les aplicacions normals, no caldrà cap especialista. Però aquesta sofisticació del *software* bàsic requerirà informàtics amb més preparació. Proporcionalment, doncs, menys gent però més preparada.

–*Els nous camps d'aplicació* o tendències previsibles on cal aplicar els coneixements adquirits en programació són:

a) Els sistemes en temps real, xarxes i sistemes distribuïts, on s'estan aplicant els resultats de la recerca en paral·lelisme. La impredictibilitat i indeterminisme en el comportament d'aquests sistemes, juntament amb la criticitat, en molts casos, del seu funcionament, l'han portat a ser un dels grans temes d'estudi. El llenguatge Ada és una de les solucions proposades, però no l'única.

b) La robòtica, o, més en concret, el *software* per a control de robots industrials. En aquest moment gairebé cada marca de robot té el seu llenguatge i es fa evident una necessitat de normalització. D'una banda, el camp és verge, i podria semblar que una proposta estàndard fóra ben rebuda. Però els industrials que usen robots ja s'han acostumat a programar els seus moviments punt a punt, a base de pitjar un botó, i no semblen massa oberts a sofisticacions, quan el control de diversos robots en paral·lel posa els mateixos problemes que els de qualsevol sistema temps-real. Un recent treball (Huber i Juan, "Mundo Electrónico", juny de 1983) ens deia que cal un llenguatge amb estructura modular expandible de manera que es pugui adaptar a diferents requeriments, que tingui una elevada capacitat d'interacció sensorial i amb perifèrics que comuniquin el robot amb el seu entorn.

c) Les noves arquitectures, alternatives al model de Von Neumann, impliquen el disseny de llenguatges basats en altres principis. En les màquines *data-flow,* per exemple, cap operació no es realitza fins que els seus operands no estan a punt, cosa que implica un paral·lelisme i una sincronització inherents. Apareixen els llenguatges *d'assignació única,* com l'ID, el LAU o el VAL (diferent del VAL dels robots; no tenen en comú més que el nom), amb fortes connexions amb els llenguatges de tipus funcional o aplicatiu.

Per finalitzar, cal esmentar un altre tema important de treball. La programació, després de tants anys d'escriure programes d'ajut a... qualsevol cosa, ha decidit mirar-se el melic i fer programes d'ajut a la construcció de programes; és a dir, sistemes que assisteixen el programador en el seu treball en l'ús de diversos mètodes. Se'n diuen *entorns de programació* (en anglès, *programming environments.*), i és una de les més fortes línies de recerca a l'actualitat.

Nota terminològica

L'únic diccionari català d'informàtica existent parteix d'un vocabulari anterior als conceptes renovadors, dels quals hem parlat a l'apartat 3. És possible, doncs, que alguns mots que he utilitzat siguin de collita pròpia. El costum a usar-los, però, em fa no saber quins són. Els treballs que, en terminologia científica i tècnica, es duen a terme a les universitats catalanes i a la Generalitat m'ajudaran a definir-los, però, de moment, espero que el lector (i el corrector!) els accepti com a eines de treball.

Pere Botella