

# **VITA CON I COMPUTER**

## **Corso di idoneità informatica**

**Dipartimento di Economia  
Università Roma 3**

©Lorenzo Seno

### **Qualche parola di presentazione**

Il corso di supporto informatico per studenti di Economia ha un carattere un po' diverso dagli altri corsi universitari. Oltre che diverso perché non è coronato da nessun tipo di esame, è diverso nelle sue finalità: esso vuole fornire gli strumenti per sviluppare una "abilità" nell'uso dei moderni computer e degli strumenti informatici, piuttosto che fornire gli strumenti teorici di comprensione dei meccanismi profondi di funzionamento delle tecnologie informatiche attuali, sia hardware che software.

Agli studenti di Economia non è necessariamente richiesto, infatti, di diventare degli esperti informatici professionali, ma piuttosto utenti esperti e intelligenti. Detto tutto questo, il corso somiglia di più all'apprendistato per la patente di guida che a un classico corso universitario. Bisogna però aggiungere che, così come per guidare un'automobile bisogna avere capito - almeno nelle linee generali - le funzioni dei vari organi e i principi di circolazione stradale, così per utilizzare correttamente un computer bisogna avere fatto lo sforzo di comprenderne la sua struttura, almeno nelle linee generali, e imparato a muoversi nel "mondo" dell'informatica. In una parola, bisogna avere immagazzinato una "cultura generale" dell'argomento. Parte fondamentale di ogni cultura è il lessico, e quindi uno degli scopi pri-

mari di quest'iniziativa di supporto e quello di insegnare i vocaboli e i relativi concetti fondamentali dell'informatica.

Questo è lo scopo di queste dispense.

Va da sé che un computer è un oggetto molto più complesso di una automobile, o di una bicicletta, e che quindi la cultura sottostante non può che essere, a sua volta, altrettanto più complessa. Oltre a ciò, c'è il fatto che l'informatica, in tutti i suoi aspetti, dottrinari e pratici, è soggetta ad una evoluzione tra le più veloci che la storia umana abbia mai conosciuto. Tutto quindi diventa obsoleto in breve tempo, non solo le apparecchiature e gli strumenti, ma talvolta anche concetti e nozioni.

Capire e imparare alcuni principi basilari serve dunque anche a fornirsi degli strumenti per aggiornare le proprie conoscenze e mantenere vive le proprie abilità nel corso del tempo.

Quello che possiamo dare, quindi, non può che essere una spinta iniziale. Queste pagine possono solo fornire una serie di spunti e un inquadramento "culturale" per una attività che gli studenti dovranno svolgere, da soli, utilizzando le strutture della sala computer della facoltà, per sviluppare compiutamente le proprie abilità. In seguito, nella vita professionale, accademica o di ricerca, dovranno largamente tenersi aggiornati da soli, o con pochi supporti: è bene lo si tenga presente, per apprezzare che tutto quanto viene oggi offerto come opportunità di apprendimento costituirà per molte persone un'occasione forse unica in tutta la vita.

## **Getting started. Elementi di igiene**

Gli attuali computer si usano con tastiera e mouse, ed è opportuno mettere in guardia da troppo facili ottimismo sulle capacità presenti e future dei sistemi di "riconoscimento della voce", i quali probabilmente non raggiungeranno mai - a dispetto di quanto recitano effimere mode o interessate manie - un livello

sufficiente di funzionalità e generalità per potere soppiantare completamente l'interazione manuale con il computer. Inutile illudersi quindi e cullarsi nell'attesa del "computer che ascolta e capisce". Tastiera e mouse (o qualcosa di molto simile) sono e resteranno a lungo il modo privilegiato di interagire con il computer. Chi non sa battere sulla tastiera con sufficiente scioltezza e rapidità deve quindi esercitarsi, se non vuole restare indietro. Chi non sa usare il mouse, deve imparare il gioco della coordinazione sguardo-polso che è alla base del suo funzionamento. Usare il mouse richiede un tipo di coordinamento simile a quello richiesto ai pistoleri (che si esercitano a sparare e colpire nel punto fissato dallo sguardo). Tutto questo può sembrare difficile a chi non si è mai cimentato, ma in realtà non lo è affatto: bastano poche ore di uso per prendere il via. L'uso di alcune accortezze può semplificare il compito, ed evitare qualche inconveniente delle prime ore.

### **Postura: come evitare le tendiniti.**

Per imparare a usare il mouse, bisogna esercitarsi a portare "di colpo" il cursore sullo schermo nel punto fissato dallo sguardo, senza guardare il mouse. Bisogna sforzarsi di tenere il polso morbido e sciolto, e a non indurire l'avambraccio. Essendo tutti noi, creature di questo millennio, destinati volenti o nolenti a passare molte ore di fronte allo schermo usando tastiera e mouse, dobbiamo imparare anche a stare attenti alle insidie "fisiche" che si possono nascondere.

Tralasciando per ora le problematiche legate alla vista e ai tubi a raggi catodici (forse in via di rapido superamento tecnologico), bisogna stare attenti agli inconvenienti in agguato legati alla postura. Il piano di lavoro tastiera-mouse non deve mai essere troppo alto, né troppo discosto dal corpo. Lavorare a braccia tese, peggio se sollevate, può provocare subdolamente delle tendiniti, difficili da curare e molto fastidiose, per buscarsi le quali non è quindi indispensabile essere dei tennisti o dei pianisti in

surmenage, ma basta anche solo essere dei disaccorti utenti dell'informatica.

La posizione ideale è indicata dai cosiddetti "tavolini dattilo" delle buone vecchie segretarie (razza ormai sparita, soppiantata appunto dai computer, non senza qualche rimpianto). I roboanti "tavoloni direzionali", alti e immensi, a parte ogni altra considerazione, sono da considerare invece un vero e proprio attentato alla incolumità fisica del tastierista.

## **Pensare = calcolare? Il ragionamento.**

I calcolatori, come dice il nome stesso, sono strumenti per il calcolo. Va detto che, se in italiano e in inglese la lingua pone l'accento su questo aspetto (*computer*, in inglese), il francese preferisce invece porre l'accento su altre capacità dello strumento: quella di ordinare e conservare le informazioni (*ordinateur*, letteralmente "ordinatore").

Il calcolo è una attività che ci è piuttosto familiare, al punto tale da essere diventata istintiva, e da essere raramente oggetto di osservazione critica.

Ciò non ostante, dietro questa parola – "calcolo" - si nascondono molte insidie e sottigliezze. Senza avere la pretesa di fornire qui una trattazione approfondita, tenteremo di osservare con occhi diversi alcune di queste attività familiari, per così dire, "dall'esterno", cominciando dalla vecchia, cara operazione del contare.

## **Il contare**

Il contare è, per la specie umana, una attività che risale ai suoi primordi. Evidenze della capacità di contare risalgono all'età della pietra. Quando si dice tre capre, o tre dinosauri (o tre pietre, o tre frecce), si mettono in relazione cose del tutto diverse, e il numero tre rappresenta questa relazione. Si tratta forse di uno dei primissimi passi verso il pensare astratto. Quando si di-

ce "tre capre e due capre uguale cinque capre", si esprime una verità del tutto indipendente dalle capre, valida per esse come per qualunque altra cosa.

Stiamo accennando all'aritmetica, come si vede, ai suoi numeri (interi), alle loro proprietà e alle operazioni su di essi.

Facciamo attenzione ora alla natura di questa "astrazione" rappresentata dal numero intero che rappresenta quanto è "popolato" un determinato insieme (la "cardinalità" dell'insieme). Essa mette in relazione insiemi di oggetti diversi, rappresentando la loro comune proprietà di essere composti da quel determinato numero di elementi. E' piuttosto evidente come le proprietà di questa astrazione non dipendano dal modo con il quale il numero è denotato. Niente cambia se scrivo tre, oppure 3, oppure ancora ///, se adotto la convenzione una barra = una unità. Una cosa è il numero, un'altra la sua rappresentazione. Nell'inventare una rappresentazione dei numeri interi, sono fondamentalmente libero di adottare qualunque convenzione, purché rispetti le caratteristiche e le proprietà dei numeri interi. Ad esempio, sarà bene che il mio sistema simbolico di rappresentazione sia in grado di rappresentare "tutti" i numeri interi. Inoltre, sarà bene che non permetta di rappresentare lo stesso numero in due modi diversi (ciò porterebbe a notevoli scomodità). Inoltre, devo "trasferire", per così dire, nel mio sistema simbolico, le regole per eseguire le operazioni fondamentali: somma, sottrazione, moltiplicazione. Più queste regole appariranno semplici, meglio sarà.

In effetti, la prima notazione numerica, storicamente parlando, si basava proprio sulla convenzione sopracitata: una barra = una unità. E' la *notazione a tacche dell'età della pietra*. Si tratta di un sistema particolarmente semplice, com'è facile intuire, e che rispetta sia la completezza che l'univocità della rappresentazione, come è altrettanto facile convincersi. Anche le regole per la somma (e la sottrazione) vi sono bene rappresentate. Per usare una terminologia moderna, la notazione è costituita da stringhe di tacche (il simbolo "/") di lunghezza (numero di tac-

che) pari al numero. La somma è una operazione simbolica piuttosto semplice: non si tratta che di giustapporre le stringhe.

$$//// + /// = //////////$$

Anche la differenza non si presenta particolarmente difficile:

$$//////// - // = ////$$

La notazione non manca però di difetti: i numeri grandi, ad esempio, occupano molto spazio, e questo è un difetto di ordine pratico. Ma c'è anche un difetto di natura più fondamentale: usando una notazione che utilizza un solo simbolo (cioè, un sistema "unario", come si dice), non siamo in grado di rappresentare esplicitamente lo zero. Lo zero denota l'assenza totale di un qualche cosa ma, come si capirà molto dopo l'età della pietra, è un numero intero come tutti gli altri, anzi, con un ruolo privilegiato.

$$//// - //// =$$

Il sistema a tacche, come abbiamo avuto occasione di dire, è "unario". Tre si dice "uno-uno-uno". Cinque si dice "uno-uno-uno-uno-uno", e così via.

Noi usiamo il sistema decimale, che è quello che ci viene insegnato alle scuole elementari. Contiamo, cioè, per unità, decine, centinaia, eccetera. Usiamo, cioè, come "base", il numero dieci e le sue potenze. Si noti che anche l'unità è una potenza di dieci:  $10^0 = 1$ . Anche il sistema a tacche utilizza una base e le sue potenze, anche se non lo si vede a colpo d'occhio: usa come base 1 e le sue potenze:  $1^0 = 1$ ,  $1^1 = 1$ ,  $1^2 = 1$ , ecc. Come si vede, la sua peculiarità (e prolissità) dipende dal fatto che, ahimè, le potenze di uno sono tutte uguali, e tutte uguali ad 1, la base stessa.

Il sistema decimale è invece logaritmico. La lunghezza delle stringhe (sequenze di caratteri) cresce logaritmicamente con la grandezza del numero, ma in compenso abbiamo bisogno di più di un simbolo base: ne servono ben dieci distinti: 0,1,2,3,4,5,6,7,8,9. Inoltre, le regole simboliche dell'addizione (ricordate? Quelle che si imparano alle elementari) sono un po' più complesse: non si tratta solo di giustapporre stringhe.

Esistono altre alternative, per la numerazione intera? Certo, non v'è nessuna ragione di privilegiare il numero dieci, se si esclude il fatto che, come specie, siamo forniti di norma di complessivamente dieci dita delle mani. La ragione dell'adozione del numero dieci è solo questa, è una ragione biologica, non matematica, legata all'uso dei primissimi "pallottolieri": le dita delle proprie mani.

E' possibile, come base per il sistema di numerazione, adottare qualunque numero intero. Una specie con due braccia e un solo dito per braccio avrebbe probabilmente adottato il numero due, come base numerica. Avrebbe cioè adottato un sistema *binario*.

Come funziona il sistema binario? Esattamente come quello decimale, salvo che invece di contare per unità, decine, centinaia, ecc., si conta per unità, duine, quartine, ottavine, sedicine, ecc. (*Attenzione!* Terminologia inventata *hinc et nunc* dal sottoscritto, non rispondente a nessuno standard). In effetti:  $2^0 = 1$ ,  $2^1 = 2$ ,  $2^2 = 4$ ,  $2^3 = 8$ ,  $2^4 = 16$  ...

Essendo il nostro sistema appunto binario, abbiamo bisogno di solo due simboli. Per usare segni noti, "0" e "1". Un numero binario è una stringa composta dai segni 0 e 1, esattamente come un numero (di base) decimale è una stringa composta dai segni: 0,1,2,3,4,5,6,7,8,9.

Come si leggono i numeri binari? In modo del tutto analogo a come si leggono quelli decimali, *mutatis mutandis*, cioè la base.

Come "si legge" 56.473? Ricordiamo dalle elementari, da destra a sinistra:

3 unità	3
7 decine	70
4 centinaia	400
6 migliaia	6000
5 decine di migliaia.	50000
	-----
	56473

Come "si legge" 10011101? Da destra a sinistra:

	in binario:	in decimale:
1 unità	1	1
0 duine	0	0
1 quartina	10	4
1 ottavina	100	8
1 sedicina	1000	16
0 trentadue	0	0
0 sessantaquattre	0	0
1 centoventottina	1000000	128
	-----	-----
	10011101	157

## Le operazioni

### Addizione

E per le somme? Ricordiamo, sempre dalle elementari, le regole per i numeri in base dieci:

$$\begin{array}{r} 1253+ \\ 0439 \\ \hline 1692 \end{array}$$

Si somma "per colonne", facendo attenzione al gioco dei "riporti" (*carry*, in inglese).

Notare che il riporto non potrà mai essere superiore a 9, e quindi influenzerà solo la colonna immediatamente più a sinistra.

Analogamente per il binario:

$$\begin{array}{r} 10010101+ \\ 0010011 \\ \hline 10101000 \end{array}$$

Notare qui il gioco dei riporti: la regola è:

$$\begin{array}{l} 0 + 0 = 0 \\ 0 + 1 = 1 \\ 1 + 1 = 0 \text{ e riporto } 1. \end{array}$$

Anche qui, il riporto non può essere mai maggiore della base, e influenza solo la colonna immediatamente a sinistra, salvo il fatto che i riporti si possono concatenare verso sinistra anche a lungo.

Tralasciamo qui un esame approfondito delle regole della moltiplicazione, che il lettore potrà, sulla base delle tracce che seguono, rivisitare da solo. Ricordiamo che la regola formale per il decimale impone l'apprendimento delle famose "tabelline", ovvero dei risultati delle moltiplicazioni tra loro delle singole cifre (digit) decimali: le tabelline dallo zero (banale) a quella del nove. Su questa base, mediante meccanismi di moltiplicazioni elementari e somme con riporto, si esegue la moltiplicazione decimale.

Analogamente per la moltiplicazione binaria. Solo che qui non abbiamo che due digit: 0 e 1, quindi le tabelline sono solo due, e poco popolate: quella dello zero (banale), e quella dell'uno (altrettanto banale):

Eccole qui:

$$0 * 0 = 0$$

$$1 * 0 = 0$$

$$0 * 1 = 0$$

$$1 * 1 = 1$$

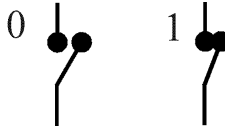
Perché tutta questa attenzione alla numerazione binaria, che non sembra poi tanto più comoda di quella unaria, a parte il fatto che ci permette, a differenza di quest'ultima, di esprimere lo zero?

E' presto detto: perché è quella usata dentro i calcolatori, e perché, come vedremo più oltre, è direttamente collegata con un altro tipo di calcolo, il *calcolo logico*.

La numerazione binaria si presta bene ad una implementazione elettrica. Porgiamo qui l'esempio degli interruttori (dentro i calcolatori si usano interruttori, anche se non meccanici, ma elettronici, basati su transistor).

Un interruttore ha due stati: "aperto" se non lascia passare corrente, "chiuso" se la lascia passare (attenzione all'inversione di significato, nel linguaggio tecnico, rispetto alla lingua ordinaria). Associamo lo stato di "aperto" (luce spenta) allo 0, e quello di "chiuso" ad 1.

La figura seguente illustra la convenzione.

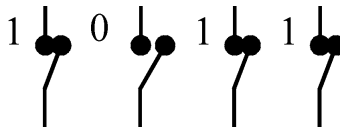


Se adesso proseguiamo la convenzione che la situazione di passaggio di corrente nel circuito corrisponde a 1, e di non passaggio a 0, anche per quello che riguarda l'uscita (supponendo cioè di considerare 1 la situazione "luce accesa" e zero "luce spenta"), abbiamo un modo semplice di realizzare varie utili funzioni sui numeri binari.

La prima è la memoria. Quando dobbiamo ricordarci un numero (di telefono, ad esempio), lo scriviamo su di un foglio. Esiste una equivalente operazione fatta con interruttori? Possiamo usare degli interruttori per segnare e "ricordare" un numero?

Sì, utilizzando la numerazione binaria. Ad esempio, con quattro interruttori possiamo ricordare i numeri da 0 a 15. Se in una stanza abbiamo tre luci, possiamo utilizzarle per "ricordare" un numero compreso tra 0 e 7. Se associamo ad ogni numero un particolare messaggio (Esempio: 0 = "Torno stasera a cena"; 1 = "Tizio si è fatto vivo e ti cerca"; 2 = "L'assassino si nasconde in cucina"; 3 = eccetera, eccetera) possiamo, come in un thrilling, scambiare informazioni con i nostri conviventi senza che nessuno se ne accorga.

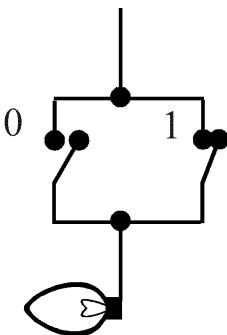
Ecco dunque una "memoria" binaria, fatta con interruttori, mentre ricorda il numero decimale 11 (1011 binario):



Ma con gli interruttori possiamo fare ben altro: possiamo cioè sommare digit binari, e farne il prodotto.

Vediamo un esempio con due soli digit. Senza entrare in dettagli elettrici, considerate da qui in poi che la lampadina si accende se esiste almeno un percorso non interrotto che la unisce al punto di partenza in alto. Si tratta di qualcosa di molto simile a quello che avviene nei veri circuiti elettrici.

Iniziamo con la somma:



E' facile convincersi che questo circuito realizza "quasi" una somma.

Infatti:

**0 + 0 = 0** (aperto e aperto = aperto, nessun percorso non interrotto, quindi luce spenta).

**0 + 1 = 1** (aperto e chiuso = chiuso, c'è almeno un percorso, quindi luce accesa).

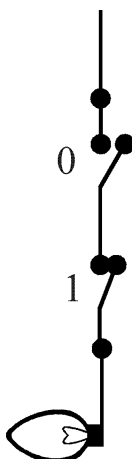
**1 + 1 = 1** (chiuso e chiuso = chiuso, ci sono ben due percorsi, luce accesa)

Dal punto di vista della somma, quest'ultimo risultato non è proprio corretto: la somma dovrebbe dare 0, con il riporto di uno. Tralasciamo quest'ultimo dettaglio, assicurando che esistono metodi, che complicherebbero però inutilmente in questa sede il

ragionamento, per modificare le cose in modo da ottenere il risultato giusto tenendo conto del resto.

## Moltiplicazione

Analizziamo ora quest'altra disposizione delle cose:



Anche qui, è facile convincersi che questa disposizione realizza la moltiplicazione di due digit binari. Infatti  $0 * 0 = 0$  (aperto e aperto = aperto),  $0 * 1 = 1 * 0 = 0$  (Aperto e chiuso = aperto).  $1 * 1 = 1$  (chiuso e chiuso = chiuso).

In conclusione abbiamo visto come, adottando un sistema di numerazione binario, sia abbastanza semplice (almeno in linea di principio) realizzare circuiti elettrici in grado di eseguire somme e moltiplicazioni.

Oltre al sistema binario e a quello decimale (e tralasciando l'antico sistema unario, che abbandoneremo d'ora in poi al suo destino), esistono altri sistemi effettivamente adottati? Sì: per ragioni di "comodità strutturale" dentro i calcolatori si usano spesso sistemi di numerazione con basi pari a potenze di due (in

particolare,  $8 = 2^3$ , che dà luogo al sistema ottale, e  $16 = 2^4$ , che dà luogo al sistema esadecimale). Se si adottano potenze di due, infatti, è facile ricondurre i digit a gruppi di digit binari. Ad esempio, il sistema ottale, che fa uso dei digit da 0 a 7, può essere pensato come un sistema binario nel quale i digit binari (che prendono il nome di *bit*, e così li chiameremo d'ora in poi), sono raggruppati a tre a tre. Analogamente per l'esadecimale, raggruppando i bit a quattro a quattro. Con questo sistema servono più cifre delle 10 che mette a disposizione il sistema decimale (occorre rappresentare con un digit i numeri da 0 a 15), quindi è necessario aggiungere altri simboli: A, B, C, D, E, F (rispettivamente per 10,11,12,13, 14 e 15).

Esempi (notare che il numero binario è lo stesso nei due casi):

**101 110 001 010**

**5 6 1 2**

(raggruppando a tre a tre, ottale)

**1011 1000 1010**

**B 8 A**

(raggruppando a quattro a quattro, esadecimale).

Un'altra base utilizzata in casi un po' particolari è 64 ( $2^6$ ). Il perché è presto detto. Avrete notato che più la base è grande, più servono digit elementari, ma più corta diventa la stringa di numeri necessaria a rappresentare i numeri.

Con una base B, una stringa di lunghezza N riesce a rappresentare i numeri compresi tra 0 e  $B^N - 1$ . Ad esempio, con 8 bit riusciamo a rappresentare tutti i numeri compresi tra 0 e  $2^8 - 1$  ( $0 \div 255$ ). Il digit formato da 8 bit prende il nome di *byte*. Quindi, in altre parole, con un byte riusciamo a rappresentare tutti i numeri interi da 0 a 255.

E con due byte? E' presto detto. Due byte sono 16 bit, quindi con essi si rappresentano i numeri tra 0 e  $2^{16} - 1$ , cioè 0 -

65535. Digit composti da un numero di bit diverso da 8 (16, ad esempio) prendono il nome di *parola* (*word*). Caso speciale la parola di 4 bit, che prende il nome di *Nibble* ( un Byte = 2 Nibble, quindi).

Con una parola da 6 bit riusciamo a rappresentare, con un solo digit, un numero compreso tra 0 e 64. Con soli otto di questi digit, riusciamo a rappresentare un numero tra 0 e  $64^8$ , pari a circa 280.000 miliardi, un numero piuttosto ragguardevole che in binario richiederebbe 512 bit, e in decimale 15 cifre. Ecco il motivo dell'uso della base 64, per trasmettere numeri grandi usando stringhe piuttosto corte. Quando si ha a disposizione un vocabolario di 64 simboli diversi, conviene usare la base 64: e infatti è questo il metodo di codifica detto appunto "base 64", utilizzato per trasmettere i file in forma compatta senza utilizzare il binario puro. Questo metodo di codifica è utilizzato attualmente nella posta Internet, per la trasmissione degli allegati. Tutto questo sarà più chiaro nel seguito.

## **Codifica e informazione.**

Tra le nozioni di bit, byte e word, e quella di "quantità di informazione", c'è uno stretto nesso. Esploriamolo un po', per precisarne meglio il significato.

Gli uomini comunicano tra loro attraverso parole e, se si escludono i portatori di lingue ideogrammatiche, le parole scritte sono composte di lettere, prese da un alfabeto (variabile da lingua a lingua, ma una volta fissata quest'ultima, perfettamente determinato). Comunicano anche attraverso i numeri, ma normalmente utilizzano questi ultimi per comunicarsi informazioni circa *le quantità*. Nel fare questo, hanno bisogno di indicare assieme al numero anche una unità di misura (a meno che non sia ovvia nel contesto). La pura affermazione "quanto prima le invierò 2,3 di patate" è palesemente destituita di significato. 2,3 di che cosa? "2,3 quintali di patate" è invece una informazione decifrabile.

Anche l'informazione verbale scritta può però essere ridotta a numeri. Basta codificare, ovvero stabilire una biunivoca corrispondenza tra lettere e numeri.

Niente ci vieta, cioè, di stabilire una numerazione delle lettere, e di utilizzare al loro posto i corrispondenti numeri. Sarebbe estremamente scomodo, ammettiamolo pure, ma ciò non ostante, possibile senza perdita di informazione.

Quello che ci serve è semplicemente di metterci d'accordo su una tabellina di codifica standard: tale lettera, tale numero. Il più diffuso di questi standard è l'ASCII (utilizzato prima nelle telescriventi, poi nei calcolatori, fino a quelli di oggi).

Ecco qui la tabellina dell'ASCII esteso in uso nei PC IBM compatibili (Windows 95 - carattere Times New Roman):

0	1	2	3	4	5	6	7	8	9	10
0	?	?	?	?	?	?	?	?	?	NL
10	?	?	CR	?	?	?	?	?	?	?
20	?	?	?	?	?	?	?	?	?	-
30	?	blank	!	"	#	\$	%	&	'	(
40	)	*	+	,	-	.	/	0	1	2
50	3	4	5	6	7	8	9	:	;	<
60	=	>	?	@	A	B	C	D	E	F
70	G	H	I	J	K	L	M	N	O	P
80	Q	R	S	T	U	V	W	X	Y	Z
90	[	\	]	^	_	`	a	b	c	d
100	e	f	g	h	i	j	k	l	m	n
110	o	p	q	r	s	t	u	v	w	x
120	y	z	{		}	~	Del	?	?	,
130	f	„	...	†	‡	^	‰	Š	‹	Œ
140	?	?	?	?	‘	’	“	”	•	–
150	—	~	™	š	›	œ	?	?	ÿ	?
160	ı	¢	£	¤	¥	¦	§	¨	©	ª
170	«	¬	-	®	¯	°	±	²	³	´
180	µ	¶	·	¸	¹	º	»	¼	½	¾
190	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç	È
200	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò
210	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü
220	Ý	Þ	ß	à	á	â	ã	ä	å	æ
230	ç	è	é	ê	ë	ì	í	î	ï	ð
240	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú
250	û	ü	ý	þ	ÿ	?				

Come si vede, sono 255 caratteri diversi, incluse le lettere italiane (accentate incluse), i digit decimali (0 ÷ 9) molte lettere straniere, segni di interpunzione, segni speciali. Le celle contenenti il simbolo ? indicano caratteri speciali, ai quali non corrisponde nessun grafismo e non sono quindi "stampabili". Quelli tra questi di codice inferiore a 32 sono utilizzati come caratteri di servizio nelle trasmissioni, ne vedremo un paio tra poco.

Un cenno a parte merita il carattere numero 32: esso è il "blank", lo spazio bianco, che è a sua volta un carattere (che denota l'assenza di ogni altro: è un po' come lo 0 per i numeri). E come per lo zero, è qui il caso di riflettere un attimo sulla sua importanza: senza spazi bianchi, in un testo tutte le parole sarebbero attaccate l'una all'altra, con quale risultante confusione è facilmente immaginabile. Quindi, codificare lo spazio bianco non è solo utile: è indispensabile ai fini della corretta decifrabilità di un testo.

Altri caratteri particolari sono il CR (Carriage Return) e NL (New Line), due esempi di caratteri "di servizio", da intendersi piuttosto come "comandi". Nelle antiche telescriventi, la ricezione di questi caratteri comportava rispettivamente il ritorno del carrello di stampa nella posizione più a sinistra del foglio (CR) oppure il salto alla riga successiva (NL), da cui i rispettivi nomi. Con questo significato si sono conservati fino ad oggi.

Di questa tabella, la prima metà (0÷127) è assolutamente standard: costituisce l'antico "ASCII standard". La seconda metà (densa di caratteri speciali e di accentate) lo è meno, e dipende dal tipo di carattere e dal tipo di sistema.

Il numero 255, infine, non è a caso. Infatti sappiamo che con un byte si esprime esattamente un numero compreso tra 0 e 255. Quindi con un byte possiamo codificare un carattere ASCII esteso. Possiamo dunque, con questo mezzo, trasformare un qualsiasi testo, purché scritto mediante questo alfabeto, in una sequenza di byte, quindi, in ultima analisi, di numeri interi binari a otto bit.

Un testo di 1500 "battute" (circa una cartella giornalistica) richiederà 1500 byte, per essere codificato. Un testo di 3000 battute, 3000 byte. Va da sé che, a parità di altre condizioni, ci si aspetta che un testo di 3000 battute contenga diciamo il doppio di informazione di uno di 1500 (a parità di altre condizioni, badate bene). Rimandando ad un testo di Teoria dell'informazione per un esame più approfondito (vi sono parecchie sottigliezze, nel concetto di informazione, legate alla probabilità e all'entropia), vogliamo solo qui fare notare che, grosso modo, il byte, così come il bit, il nibble e ogni altro digit possono fungere, attraverso questo meccanismo, da unità di misura dell'informazione.

In informatica l'unità fondamentale è il byte, proprio a causa della codifica ASCII che, come è facile intendere, riveste una importanza fondamentale.

Come ogni unità di misura, c'è però bisogno di multipli (i sottomultipli ce li abbiamo già: Nibble e bit). Quanti byte sono necessari per codificare l'Enciclopedia Treccani?

I multipli, analogamente a tutti gli altri sistemi di misura, sono il K (Kilo), Mega, Giga, Tera (byte). Solo non dimentichiamo che il sistema utilizzato è binario, *non decimale*, quindi:

<b>1 K = 1024</b>	<b>(non 1000!)</b>
<b>1 M = 1 K * 1 K</b>	<b>non 1.000.000 ma 1.048.576</b>
<b>1 G = 1 K * 1 M</b>	<b>non 1.000.000.000 ma 1.073.741.824</b>
<b>1 T = 1 K * 1 G</b>	<b>non 1.000.000.000.000 ma 1.099.511.627.776</b>

## **Il ragionamento e la logica. Il "calcolo logico".**

Abbiamo visto come si possano fare abbastanza facilmente i calcoli dell'aritmetica utilizzando circuiti elettrici, grazie alla numerazione binaria. Tutto si basa sul fatto che i calcoli dell'aritmetica sono basati su regole ferree, perfettamente specificate in modo non ambiguo, e, notiamo ancora, (se ci si limita alla so-

la aritmetica), in modo "finitistico", usando cioè descrizioni delle regole di lunghezza finita, utilizzando tutt'al più il concetto di "successivo" (che, come noto, dà luogo ad una sequenza di interi senza fine).

Ma che dire del "ragionamento"? Il ragionamento matematico, scientifico (ma anche filosofico) sottintende un concetto di rigore, una "logica", una qualche forma di metodo che legittimi la correttezza dei risultati di un ragionamento.

Di questo problema la specie umana si è interessata fin dall'antichità classica. Le prime forme di "logica formale", di "scienza della deduzione" risalgono ad Aristotele e agli Stoici. Non c'è disciplina del pensiero che non abbia il problema della sua logica, dei suoi metodi, e delle garanzie di correttezza che da questi metodi derivano. In nessun campo sarebbe accettato un ragionamento "privo di ogni logica".

Un particolare rilievo nello sviluppo di una logica formale ha assunto storicamente, in epoca più recente, la matematica, a causa dell'enorme sviluppo che ha conosciuto tra il 700 e l'800, al conseguente moltiplicarsi di metodi di dimostrazione, e anche alla scoperta di errori e antinomie nelle teorie che si andavano sviluppando. Già Leibniz (1646-1716), con la sua idea di un "calculus ratiocinator" aveva vaticinato quello che oggi si chiama "calcolo logico". In epoca recente, la storia della logica e della critica dei fondamenti della matematica, per certi aspetti anche drammatica (Frege si suicidò in seguito alla scoperta della cosiddetta "antinomia del mentitore" nella sua teoria degli insiemi) ha conosciuto uno sviluppo sconvolgente grazie, oltre a Gottlob Frege (1846-1925), a Bertrand Russell e Whitehead, a Boole (1815-1864), Peano, Zermelo, Hilbert, Gödel, per non citare che alcuni.

Proviamo qui ad affrontare gli elementi base di una logica delle proposizioni, appoggiandoci alla "algebra della logica" di Boole. Una proposizione è una affermazione linguisticamente compiuta relativa ad una qualunque entità pensabile. Chiamiamola "p". Accettiamo senza discutere (anche se c'è chi lo discute) il prin-

cipio del *tertium non datur*, ovvero che questa affermazione possa essere disgiuntivamente vera oppure falsa, senza altre possibilità (in particolare, escludiamo che non possa essere né vera né falsa).

Un esempio? Ecco il classico "tutti i corvi sono neri". Il valore di verità della proposizione può dunque, in questo contesto, assumere solo due valori: Vero o Falso (cioè, può assumere un valore *binario*).

Qualcuno obietterebbe qualcosa di simile: I corvi sono tanti (se invece di corvi, si trattasse di entità matematiche, potrebbero essere addirittura infiniti). Non è affatto detto che sia facile decidere se effettivamente tutti i corvi siano neri. Potrebbe trattarsi quindi di una proposizione né vera né falsa, ma piuttosto *indecidibile*. Quindi, risulta necessario introdurre un altro valore di verità, accanto ai tradizionali "vero" e "falso": e cioè "indecidibile". Quella che serve, quindi, è una logica con tre valori di verità. Qualcun altro obietta che tra vero e falso vi sono tutte le sfumature dell'incerto, propugnando una logica della probabilità (la *fuzzy logic*).

Noi trascureremo però queste strade, e ci occuperemo qui solo della logica del *tertium non datur*, della logica del Vero e del Falso. Per brevità indicheremo questi valori con V e F. Non preoccupiamoci troppo della corrispondenza alla realtà di questi valori (cioè, se la proposizione p sia effettivamente, contenutisticamente, vera o falsa). Siamo alla ricerca di metodi e "verità" formali, indipendenti cioè dal contenuto delle affermazioni.

Quali operazioni possiamo fare, relativamente al valore di verità di una sola proposizione? Dato che i valori di verità sono solo due, è presto fatto il conto di tutte le possibilità: la proposizione di partenza può essere V o F, e così quella d'arrivo. Vediamo tutte le possibilità, che non sono che 4. Pensiamo  $p_i = O_i(p)$ , cioè la proposizione d'arrivo come risultato di una operazione applicata alla proposizione di partenza, che la trasforma:

p	p <sub>1</sub>	p <sub>2</sub>	p <sub>3</sub>	p <sub>4</sub>
V	V	V	F	F
F	V	F	V	F

La prima delle operazioni (così come l'ultima) non ha alcun interesse: essa "forza" a Vero (Falso) il risultato finale, indipendentemente dalla proposizione di partenza. La seconda anche è priva di interesse: lascia le cose come stanno, ovvero è l'identità. Solo la terza è utile: essa rovescia il valore di verità della proposizione di partenza. Se era Vera, la trasforma in Falsa, e viceversa. Questa operazione "monoargomentale" (agente cioè su una sola variabile, intesa come proposizione), è chiamata (non a caso) *negazione* e viene indicata con il segno  $\neg$  che precede la proposizione:  $\neg p$  (leggi: non-p). Dunque la negazione, *per definizione*, rende vera una proposizione falsa, e falsa una vera (non diversamente da come si intende usualmente).

Dal punto di vista dell'articolazione sintattica della proposizione, a cosa corrisponde l'operazione di negazione? Se da punto di vista meramente formale, operatoriale, basta far precedere alla variabile il segno  $\neg$ , dal punto di vista sintattico non sempre basta inserire un "non" in un punto qualunque dentro la proposizione, a precedere magari il verbo.

Se la proposizione è un'asserzione singolare, fattuale (per così dire) come "a è nero" (con a singolare, non una classe), è abbastanza facile negarla: "a è non nero" (forma più precisa della usuale: "a non è nero"). Ma se l'affermazione ha un carattere di universalità, occorre fare una certa attenzione. La negazione di "Tutti i corvi sono neri" non è "Tutti i corvi non sono neri", quanto invece "Non tutti i corvi sono neri".

Un chiarimento essenziale, a questo proposito, lo introducono i cosiddetti "quantificatori" della logica. Essi sono due:

- " che si legge "Qualunque sia", quantificatore universale.

§ che si legge "Esiste almeno un", quantificatore esistenziale.

La formulazione delle proposizioni universali utilizzando questi quantificatori consente una gestione rigorosa e non ambigua delle negazioni, delle proprietà (universali) e dell'esistenza di entità.

Riformuliamo la proposizione  $p$  e la sua negazione facendo uso dei quantificatori:

*Affermazione:*

"Tutti i corvi sono neri", in modo informale.

"Qualunque sia  $a$ , se  $a$  è un corvo,  $a$  è nero",  $p$  formalmente espressa.

*Negazione:*

"Esiste almeno un  $a$ , con  $a$  corvo, che è non nero",  $\neg p$ .

"Esiste almeno un corvo non nero", negazione informale (ma corretta).

Ciò detto, domandiamoci ora cosa possa succedere "combinando" in vari modi due proposizioni  $p_1$  e  $p_2$ . Chiediamoci cioè quali e quanti possibili "connettivi biargomentali" esistano. Procediamo sulla scorta di quanto abbiamo fatto con quelli monoargomentali, salvo il fatto che adesso le combinazioni possibile sono molte di più (per la precisione, 16).

Se si scartano quelli banali e non significativi, e quelli che si ottengono semplicemente per negazione di un altro, ne restano, di importanza fondamentale, quattro:

$p_1$	$p_2$	$p_1 \vee p_2$	$p_1 \wedge p_2$	$p_1 \supset p_2$	$p_1 \hat{\cup} p_2$
V	V	V	V	V	V
V	F	V	F	F	F
F	V	V	F	V	F
F	F	F	F	F	V

Abbiamo usato il simbolo  $\vee$  per il primo. Esso sta per il latino "vel" ("o", ma non disgiuntivo). Il simbolo rovesciato  $\wedge$  indica il connettivo "e" (congiunzione). Questi connettivi corrispondono all'uso corrente nella lingua, con una precisazione per la congiunzione "o". Nella lingua italiana esiste una sola "o", mentre in latino esisteva sia il "vel" che lo "aut". Aut indica un "o" disgiuntivo, nel quale cioè è falso il risultato della combinazione di due proposizioni vere, restando vero solo il risultato della combinazione di due proposizioni di cui una è vera e l'altra falsa. In italiano non esiste un modo formale di riprodurre questa distinzione (che i latini avevano invece chiarissima), la si può rendere con "o l'una, oppure l'altra".

La tabellina delle verità dell'aut sarebbe la seguente:

$p_1$	$p_2$	$p_1 \text{ aut } p_2$
V	V	F
V	F	V
F	V	V
F	F	F

Aut non è però un connettivo elementare, ma lo si può ottenere da una combinazione di *vel*, di *e*, e di negazioni.

Restano le ultime due colonne ( $\Rightarrow$ ,  $\Leftrightarrow$ ). Il primo dei due operatori è l'implicazione, il secondo la doppia implicazione (le condizioni necessarie, e necessarie e sufficienti, della matematica). Essi corrispondono al "se .... allora" e "se e solo se" della lingua italiana, e non sono elementari, nel senso che si possono ottenere a partire da una opportuna combinazione di OR, AND e NOT, ma restano evidenziati a causa della loro (ovvia) importanza.

Proviamo a fare qualche semplice esempio di implicazione semplice, commentando le relative tabelline di verità.:

"n è divisibile per 4 implica che n è divisibile per due" (implicazione semplice, unidirezionale. Non vale infatti il viceversa).

$$"p_1 \supset p_2"$$

È vera: se  $p_1$  è vera e  $p_2$  è vera, e assieme se  $p_1$  è falsa e  $p_2$  è vera (Noi diciamo che la suddetta implicazione è valida, perché qualunque sia n, o esso è divisibile per 4 e per due, oppure non è divisibile per 4, ma lo è per due). Notare come quest'ultima "asimmetria" sia indispensabile per distinguere la implicazione semplice dalla doppia implicazione.

È falsa: se  $p_1$  è vera e  $p_2$  è falsa, e assieme se  $p_1$  e  $p_2$  sono false ("Se n è divisibile per 2, allora n è divisibile per 4" è falsa, come noto, così come lo è "se n non è divisibile per 4, allora n non è divisibile per 2").

Si può facilmente verificare che questi due periodi non fanno altro che riferire in lingua italiana corrente quanto molto concisamente specificato nella tabellina di verità di  $p_1 \Rightarrow p_2$ .

Questo insieme di condizioni di verità e falsità definisce il rapporto di implicazione semplice, nel senso comune del termine, come abbiamo potuto verificare.

Potremmo pensare a questo punto a come siano fatti i connettivi tri, quadri, penta, ecc. argomentali (e al loro significato). Essi sono però tutti ottenibili a partire da quelli biargomentali "vel", "e", e da quello monoargomentale "negazione". Quindi non c'è motivo di introdurli con simboli a parte.

Come si può notare, stiamo costruendo una "algebra della logica". Le tabelline indicano le regole per effettuare le operazioni, e quindi, una volta ammesso l'uso delle parentesi e, per comodità, una regola di precedenza degli operatori (esattamente come si fa in algebra), siamo in grado di sviluppare calcoli logico-deduttivi e di stabilire dei teoremi (esattamente come si fa in algebra), ovvero "verità universali", indipendenti dal valore che assumono le variabili.

Ad esempio, la frase "Se piove oppure nevicata, e non nevicata, allora piove" è una frase forse priva di una sua qualche utilità pratica, ma indubbiamente vera sempre, indipendentemente dal fatto che piove, nevichi, o ci sia un bel sole. Essa è un "teorema" (ovvero una tautologia) esattamente nello stesso modo in cui è un teorema  $(a + b)^2 = (a^2 + b^2 + 2ab)$ , la quale è una affermazione sempre valida, indipendentemente dai valori di a e b.

L'espressione logico-algebrica della proposizione sulla pioggia è la seguente:

$$((p1 \dot{\cup} p2) \dot{\cup} (\emptyset p2)) \supset p1$$

Applicano le tabelline, nell'ordine (dall'interno verso l'esterno) stabilito dalle parentesi, esattamente come in algebra, per tutte e quattro le combinazioni di valori di verità di p1 e p2, ci si convince facilmente come il risultato finale, il valore dell'intera pro-

posizione sopra enunciata, è sempre Vero, indipendentemente dai valori di verità delle proposizioni di partenza.

Potete facilmente riottenere la proposizione di partenza, semplicemente operando le sostituzioni:

$p_1 \rightarrow$  "piove"

$p_2 \rightarrow$  "neveca"

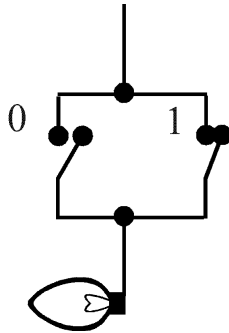
Per questa strada, si ricostruiscono tutti i "teoremi" (o regole) della logica classica, come ad esempio il sillogismo Aristotelico.

L'algebra del calcolo logico delle proposizioni alla quale stiamo accennando è l'algebra di Boole. Nel linguaggio dell'elettronica e dell'informatica si usano per i connettivi le seguenti denominazioni: AND, OR e NOT. L'aut prende il nome di "Or esclusivo", o XOR. Sarebbe facile dimostrare che questi operatori sono più che sufficienti al calcolo logico (in realtà, ne bastano due soli: ad esempio, la negazione e lo OR, con opportune combinazioni dei quali si possono ricostruire lo AND, lo AUT, l'implicazione semplice e doppia, ecc).

Abbiamo fatto qualche passo avanti, nella comprensione di come il ragionamento possa diventare calcolo (sia esso eseguito da un calcolatore elettronico oppure umano). Inoltre, speriamo di avere fornito qualche strumento per un pensiero rigoroso (che non è certo utile solo in matematica) e per una maggiore comprensione di cosa siano le deduzioni (che non si effettuano solo in matematica).

C'è però dell'altro.

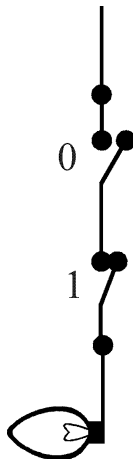
Assumiamo la convenzione che  $V = 1$  e  $F = 0$ . Osserviamo il seguente circuito:



E' facile convincersi che realizza un "vel", o OR che dir si voglia. Infatti:

<b>Aperto e Aperto = Aperto</b>	<b>(Falso o Falso = Falso)</b>
<b>Aperto e Chiuso = Chiuso</b>	<b>(Falso o Vero = Vero)</b>
<b>Chiuso e Chiuso = Chiuso</b>	<b>(Vero o Vero = Vero)</b>

Non è ancora finita: Osserviamo quest'altro circuito:



Questa volta è facile convincersi che questo circuito realizza un "e". Infatti:

<b>Aperto e Aperto = Aperto</b>	<b>(Falso e Falso = Falso)</b>
<b>Aperto e Chiuso = Aperto</b>	<b>(Falso e Vero = Falso)</b>
<b>Chiuso e Chiuso = Chiuso</b>	<b>(Vero e Vero = Vero)</b>

Se adesso confrontiamo questi due circuiti con quelli di somma e moltiplicazione, osserviamo che sono identici. Dunque con gli stessi circuiti si possono sia fare calcoli aritmetici, che *calcoli logici*.

Rimane solo da notare che il secondo dei circuiti realizza esattamente sia la moltiplicazione binaria che l'operazione logica AND, mentre il primo realizza esattamente l'operazione logica OR, e solo approssimativamente la somma.

Questo porta ad identificare le operazioni AND e \*, e (con riserva) OR e +. Si dice talvolta che AND è un "prodotto logico", mentre OR è una "somma logica".

Dato che, come già affermato, gli operatori di implicazione e doppia implicazione non sono elementari, ma possono essere ottenuti da una opportuna combinazione di AND, OR e NOT, abbiamo qui la base circuitale per eseguire anche il calcolo logico.

## Calcolo logico e insiemi

Rimane da esplorare ancora un nesso, importate ai fini anche pratici. E' quello tra calcolo logico, connettivi logici, e insiemistica. Limitiamoci qui a considerare insiemi *finiti* (un abile trucco per scansare una infinità di questioni sottili, alle quali accenneremo brevemente nel seguito).

Un insieme è definito da una proposizione relativa agli elementi dello stesso. Consideriamo per esempio l'universo (finito) delle automobili. Prendiamo in questo universo un esempio: l'insieme

X delle automobili rosse. Prendiamo, alla buona, la definizione dell'insieme come segue:

$p = "a \text{ è un elemento dell'insieme } X \text{ se e solo se } a \text{ è rossa}."$  ( $a \in X \Leftrightarrow a \text{ è rossa}$ ).

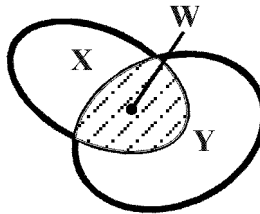
Consideriamo un altro insieme: quello Y delle automobili a benzina.

$p_2 = "a \text{ è un elemento dell'insieme } Y \text{ se e solo se } a \text{ è a benzina}."$  ( $a \in Y \Leftrightarrow a \text{ è a benzina}$ ).

Dalla teoria elementare degli insiemi sappiamo che è definita tra due insiemi l'operazione di intersezione, ovvero la costruzione di quell'insieme W che contiene gli elementi comuni ai due (cioè, nel nostro caso, le automobili rosse a benzina) . Esso si indica con:

$$W = X \cap Y.$$

Usando i diagrammi di Venn tipici dell'insiemistica:



Qual è la proposizione che definisce questo nuovo insieme (anche se costituito da "vecchi" elementi?)

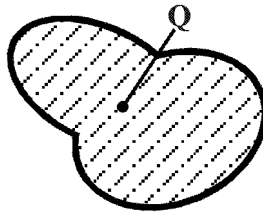
$p_4 = "a \text{ è un elemento dell'insieme } Y \text{ se e solo se } a \text{ è rossa AND } a \text{ è a benzina}."$

Notare che:  $p_4 = p \wedge p_1$ . Dunque, all'operazione insiemistica di intersezione corrisponde l'operazione booleana  $\wedge$ . Questo è la ragione per cui quest'ultima è anche detta *intersezione logica*. In forma più rigorosa, per definizione:

$$(a \in (X \cap Y)) \cup ((a \in X) \cup (a \in Y))$$

Un'altra operazione tra due insiemi è l'unione, cioè quell'insieme  $Q$  costituito dagli elementi dell'uno e dell'altro, presi ciascuno una sola volta (ricorderete come in insiemistica non si ammettano "copie" di un elemento: se un elemento compare "due volte", esso conta per una sola. Un elemento è sempre singolare).

Si scrive:  $Q = X \cup Y$ .



Qual è la proposizione che definisce  $Q$ ?

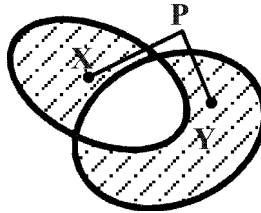
$p_5 =$  "a è un elemento dell'insieme  $Q$  se e solo se a è rossa OR a è a benzina".

Notare che  $p_5 = p \vee p_1$ . Dunque l'unione insiemistica corrisponde all'operazione booleana *vel* (OR). Questo è il motivo per il quale quest'ultima è anche detta *unione logica*. In forma più rigorosa, per definizione:

$$(a \in (X \cap Y)) \cup ((a \in X) \cup (a \in Y))$$

Domandiamoci infine, per curiosità, a quale operazione logica corrisponda l'insieme indicato nella figura seguente (potremmo scrivere:  $P = Q - W$ , intendendo con questo che esso è composto dagli elementi dell'unione di  $X$  e  $Y$ , *meno* quelli dell'intersezione tra  $X$  e  $Y$ ). Essi sono quelle automobili che sono o rosse, oppure a benzina (disgiuntivamente, *aut*). Si intendono quindi

escluse quelle che sono e rosse, e contemporaneamente a benzina.



$p_6 = "a \text{ è un elemento dell'insieme } P \text{ se e solo se } a \text{ è rossa AUT } a \text{ è a benzina}"$ .

Quindi,  $p_6 = p \text{ aut } p_1$ .

Un'ultima (utile) nota: definiamo "cardinalità" di un insieme il numero dei suoi elementi (quando essi siano finiti, come nel nostro caso). Indichiamola con  $N(A)$ , se  $A$  è l'insieme.

Sarebbe facile dimostrare che:

$N(X \cap Y) \leq \text{Min}(N(X), N(Y))$ . Il numero di elementi dell'insieme intersezione è non maggiore del più piccolo dei due. Dunque lo AND, o intersezione, *restringe* (per così dire) gli insiemi.

$N(X \cup Y) \leq N(X) + N(Y)$ , e  $N(X \cup Y) \geq \text{Max}(N(X), N(Y))$ . Il numero di elementi dell'insieme intersezione è non maggiore della somma dei due e non minore del maggiore dei due. Dunque lo OR, o unione, *espande* gli insiemi, o quanto meno non li restringe.

Torneranno utili, queste osservazioni, quando si parlerà dei sistemi per la ricerca di informazioni.

## **Il programma finitistico di Hilbert, il teorema di Gödel e il fallimento dell'idea che il pensiero non è che calcolo.**

Per quel che riguarda l'algebra della logica, o se volete il "calculus ratiocinator" vaticinato da Leibnitz, ci fermiamo a questi brevi cenni, peraltro utili in molte circostanze del ragionare e del vivere, con o senza computer. E' evidente che abbiamo finora definito solo il punto di partenza di un cammino che porta assieme alla formalizzazione delle teorie matematiche e della logica sottostante (cioè, del metodo di deduzione dei teoremi). Senza entrare in dettagli, aggiungendo alla logica delle proposizioni regole formali per la definizione dei concetti specifici delle teorie, degli assiomi e delle regole di deduzione, si ottiene la Logica dei predicati, base per una teoria generalizzata della dimostrazione e del tentativo di ricondurre la deduzione matematica ad un unico calcolo logico.

Torniamo però alla storia. Lo studio della logica e dei fondamenti della matematica alla ricerca del "rigore", inteso come garanzia di esattezza delle conclusioni del ragionamento; la necessità di riformulare teorie base (come quella degli insiemi) a causa di antinomie (paradossi, ovvero la deducibilità all'interno di una teoria sia di  $p$  che di  $\neg p$ !) scoperte successivamente, tutte queste considerazioni portarono Hilbert alla formulazione di un programma che avrebbe dovuto, nelle intenzioni, garantire la costruzione di un metodo di ragionamento (in termini più tecnici, *ad una teoria della dimostrazione*) oggettivo, neutro, autofondato, e pertanto infallibile, basato sulla logica dei predicati, e identificabile come un calcolo logico. Questo è il programma finitistico di Hilbert, il tentativo (o se vogliamo, l'illusione) di potere ottenere una ricostruzione rigorosa della matematica a partire da assiomi e regole di deduzione, intese come sequenze di stringhe finite manipolate con regole di elaborazione esattamente specificate mediante un numero a suo volta finito di simboli. Dietro questa idea c'è l'espulsione dalla matematica del concetto di "infinito attuale", e l'abolizione dei metodi di dimostrazione

(e ragionamento) che ne implicano l'uso (come ad esempio l'induzione transfinita). Questo programma, che ha raccolto molte energie nei primi anni (e raccoglie tuttora molti inconsapevoli sostenitori) si è definitivamente infranto negli anni '30 per opera di Gödel e del suo *teorema di incompletezza*, che dimostrò come per questa via non era possibile ottenere non diciamo la ricostruzione della intera matematica, ma nemmeno di una sua parte piuttosto elementare (e basilare): l'aritmetica. Egli dimostrò infatti che la produzione di teoremi dell'aritmetica per via finitistica avrebbe lasciato fuori *infiniti teoremi*, ovvero non avrebbe prodotto infinite asserzioni peraltro vere relative ai numeri interi.

Il teorema di Gödel è stato un punto di svolta nel pensiero matematico del nostro secolo. Da lì hanno preso le mosse molti tentativi di precisare il senso e la portata del suo risultato. Senza entrare nel merito di una materia complessa e specialistica, vale la pena di rendere esplicito il rapporto con il computing.

Cosa c'entrano infatti con tutto questo i calcolatori? Né all'epoca di Leibnitz, né a quella di Hilbert i calcolatori erano "tecnicamente" pensabili. Lo erano però in linea di principio, e i computer erano creature largamente immaginate (e sognate) da coloro che si occupavano di matematica sia applicata (come strumento per eseguire calcoli) che fondamentale (come strumento di verifica, o come abbiamo visto, addirittura di produzione dei ragionamenti). Il calcolatore è una macchina *finita*, finite sono le sue risorse (per quanto espandibili, con il tempo, abbastanza a piacere), così come la sua memoria. Così come finite sono tutte le cose che l'uomo può costruire (non altrettanto si può dire delle cose che l'uomo può pensare, invece). Se fosse mai stato possibile racchiudere tutto il ragionamento matematico nell'orizzonte finitistico di Hilbert, esso sarebbe stato eseguibile mediante una macchina, in modo analogo a quanto già si faceva con le calcolatrici aritmetiche meccaniche (le quali anche esse applicano regole finitistiche per seguire le operazioni). Mediante una simile macchina sarebbe stato possibile riottenere tutti i te-

oremi della matematica, quindi ricostruire tutta la matematica, e senza possibilità di errore.

In altre parole, il pensiero matematico (ma forse, a questo punto, il pensiero umano *tout court*) sarebbe stato equivalente a quanto un programma di calcolo per calcolatori può ottenere.

Così non è, come abbiamo detto, il che porta a dire che il pensiero umano (ma forse, anche quello di una seppia la quale, per quanto dotata di un sistema nervoso rudimentale, esibisce un comportamento di attacco, di difesa, di riproduzione, estremamente complesso) non è riconducibile ad un calcolo, ma necessita dell'introduzione di procedimenti tipicamente "mentali", implicanti ad esempio l'accettazione dell'idea di "infinito in atto", platonicamente esistente in sé, non inteso come un limite che si sposta ininterrottamente, ma come una entità matematica esistente con le sue proprietà e le sue regole di manipolazione. Oppure l'uso di analogie, l'identificazione di similitudini di forme, ecc.. Procedimenti cioè non formalizzabili, nel senso appena precisato di "calcolabili".

Dunque, l'affermazione che tutta la matematica è derivabile finitisticamente va negata. Come abbiamo appreso a proposito di come si effettua correttamente una negazione, ciò non significa che tutta la matematica non è derivabile finitisticamente, ma piuttosto che esistono parti della matematica (almeno una) così non derivabili. Delimitando fortemente il contesto, e accettando opportune ipotesi "indimostrabili", è possibile derivare teoremi in modo automatico, e ciò è stato effettivamente fatto utilizzando calcolatori, proprio nel quadro delle ricerche tese a valutare e precisare i contorni delle conseguenze del teorema di Gödel di cui parlavamo. In questo modo si ottengono, e a prezzo di una arbitrarietà iniziale *che implica la conoscenza del resto della matematica*, solo frammenti di teorie.

L'argomento affrontato qui, per quanto importantissimo per l'informatica (e per il pensiero umano in generale) non è dei più semplici. A chi volesse approfondire l'argomento consiglio la seguente lettura:

Douglas R. Hofstadter, *Gödel, Escher, Bach: un'Eterna Ghirlanda Brillante*. Adelphi - Milano 1990. Il libro parla del lavoro del logico-matematico Gödel, di quello del pittore Escher e di quello del compositore Johan Sebastian Bach, senza trascurare Achille, la Tartaruga, il Granchio, i fonografi rumorosi e tutta una vasta corte di altri divertenti personaggi. Si tratta di un testo di grandi qualità letterarie e dalla lettura piacevolissima (in America ha vinto il Pulitzer, che è un premio giornalistico). Non ci si lasci però ingannare: il libro tratta, sia pure in modo piacevole e spiritoso, questioni molto profonde la cui comprensione richiede comunque un notevole impegno da parte di chi legge.

## Algoritmi, software e hardware

Nel gergo dell'informatica, un insieme, finito, di regole finite, oggettive, prive di ambiguità e perfettamente specificate (al punto da non richiedere nessuna "interpretazione" e nessun esercizio di libero arbitrio da parte dell'ipotetico esecutore delle regole) prende il nome di *algoritmo*.

I calcolatori sono macchine (insiemi furbamente correlati di "interruttori" come abbiamo già visto) in grado di eseguire algoritmi, non diversamente da come potrebbe fare un operatore umano (che mettesse da parte ogni sua dote di inventiva e ogni iniziativa personale), senza però né sbagliarsi, né distrarsi, e a velocità sovrumana.

Come avremo modo di comprendere meglio nel seguito, il "compito" che un calcolatore effettua non è "scritto" nei suoi circuiti. I suoi circuiti, per come sono connessi tra loro, sono solo "predisposti" ad eseguire genericamente determinati tipi di compiti, che vengono esattamente specificati dallo stato dei suoi interruttori impartito inizialmente dall'operatore umano. Come nei circuiti di somma e moltiplicazione, ad esempio, per i quali *quale* somma venga effettuata non è specificato nel circuito (che è genericamente in grado di effettuare *qualsiasi* somma), ma piuttosto nello stato prescelto degli interruttori, che indica *quali* numeri si desidera sommare.

I calcolatori sono dunque in grado di effettuare, né più né meno, qualunque compito sia eseguibile mediante un algoritmo. Dunque i calcolatori soffrono di una prima serie di limitazioni: le stesse che hanno gli algoritmi (i quali, come abbiamo visto, non racchiudono il pensiero). Inoltre, soffrono delle loro specifiche limitazioni in quanto esecutori di algoritmi: ad esempio, un algoritmo potrebbe essere eseguibile in un tempo finito, ma essere così complesso (essere cioè composto da un numero talmente alto di operazioni elementari) da richiedere tempi di calcolo elevatissimi, fuori della portata pratica o addirittura umana. Questo secondo tipo di limite, inoltre, potrebbe essere solo temporaneo (cioè, legato all'attuale velocità dei calcolatori). Potrebbe invece essere un limite di lunga durata (bisogna ipotizzare un grande, per quanto realizzabile, progresso futuro nelle velocità di calcolo perché un algoritmo diventi utilizzabile). Potrebbe essere infine un limite tecnicamente insormontabile: anche pensando al calcolatore più veloce possibile (anche se irrealizzabile per ragioni pratiche), i tempi di calcolo potrebbero restare del tutto sovrumani.

Esistono algoritmi siffatti? E hanno importanza pratica?

La risposta è sì. Non solo, ma vi sono in realtà *problemi* (ovvero ciò che un algoritmo deve risolvere) che sono fatti così, intendendo con questo che l'insolubilità sembra più legata alla natura stessa del problema, che all'algoritmo prescelto (in generale, un dato problema può avere più algoritmi risolvitori, e tra questi, alcuni possono essere stupidi e inefficienti, altri rapidi e efficienti).

Uno di questi è il problema del commesso viaggiatore: date  $N$  città connesse tra loro da un reticolo di strade, trovare il percorso minimo che permette di visitare tutte le città almeno una volta. L'unico algoritmo che si conosce (e forse, non ne esistono altri) è il più stupido possibile: consiste nel "provare" tutti i percorsi, annotando la distanza totale, e dopo averli provati tutti, scegliere quello (o quelli) di percorso minimo. Non si conoscono "scorciatoie" a questo metodo da "forza bruta". Al crescere di  $N$ ,

il numero di percorsi che devono essere visitati cresce esponenzialmente, il che significa che molto presto il numero di percorsi supera qualsiasi cifra immaginabile, e l'algoritmo diviene impraticabile per qualunque "velocità di visitazione" <sup>1</sup>.

I problemi irriducibilmente esponenziali prendono il nome di NP-Completi (NP sta per Non Polinomiali, appunto perché esponenziali). Lungi dal costituire un dramma, si è trovato il modo di sfruttarli. Algoritmi NP-completi sono la base dei sistemi di encrittazione in chiave pubblica, i quali permettono la "firma elettronica".

Prima di andare avanti, una questione di nomenclatura: quando un algoritmo viene codificato (tradotto) in una forma adeguata per potere essere dato in pasto ad un calcolatore, prende il nome di "programma". Concettualmente, il programma corrisponde alla configurazione di aperto-chiuso degli innumerevoli interruttori di cui il computer è composto: esso è dunque un insieme di informazioni (binarie: 0-1, aperto-chiuso). Un algoritmo, un programma, esprime dunque un calcolo in senso astratto. Lo stesso algoritmo può essere programmato per essere usato su computer diversi, dando luogo a programmi diversi, così come lo stesso computer può essere programmato con algoritmi diversi, per eseguire compiti diversi.

E' questo il concetto di indipendenza tra il *software* (materia morbida) e l'*hardware* (materia dura, ma in inglese letteralmente "ferramenta"). I programmi, le informazioni, costituiscono il mondo software. I calcolatori costituiscono l'hardware.

---

<sup>1</sup> Va detto che una recente "scappatoia" è stata individuata nel cosiddetto DNA Computing, nel quale anziché circuiti elettrici, come supporto per il calcolo si usano frammenti di DNA. Questo approccio, dal valore probabilmente solo teorico, permette di superare l'esplosione esponenziale dei problemi NP.

## **Illusioni più recenti. Intelligenza Artificiale, Sistemi esperti, reti neurali, riconoscimento del parlato ...**

Dunque, con i calcolatori non solo non è possibile emulare il pensiero, ma si incontrano difficoltà di vario tipo e di vari livelli anche per l'esecuzione di compiti concettualmente più semplici del pensare. L'affermazione potrebbe sembrare banale, se non fosse per i tentativi (talvolta reiterazione di antichi tentativi) di accreditare ai calcolatori potenzialità inesistenti. Parliamo di una "Intelligenza Artificiale" che avrebbe dovuto essere incorporata nei "Sistemi esperti", programmi di calcolo cioè, sedicentamente in grado di "emulare" il ragionamento complesso effettuato da esperti umani (di volta in volta: in prospezioni petrolifere, nella diagnostica dei guasti di apparecchiature complesse, addirittura nella diagnostica medica, nella previsione degli andamenti di borsa, e così via).

I relativi programmi di ricerca (e di sfruttamento commerciale) sono sfociati regolarmente in fallimenti. Ciò nonostante analoghi tentativi vengano periodicamente riproposti sotto altre chiavi.

Di questi tentativi fanno parte le cosiddette "reti neurali", ovvero sistemi elettronici che sarebbero basati, a dire dei loro mentori, sulla imitazione del funzionamento del cervello (da cui il nome di "reti di neuroni"). Inutile dire come, ad onta del nome, i neuroni elettronici abbiano poco a che fare con quelli della fisiologia, e come ancor meno le loro reti somiglino agli strati cerebrali che pretenderebbero di imitare, seppure alla lontana. Secondo concezioni recenti di fisiologia del cervello, in realtà il neurone sarebbe assai meno importante di quanto pensato fino ad ora, mentre è proprio la rete di connessione, la sua topologia e la sua dinamica (le connessioni neuronali si muovono più o meno velocemente e la rete di conseguenza si riconfigura durante tutto l'arco della vita) a essere probabilmente importante ai fini di quello che noi chiamiamo "pensiero".

Altri fallimentari e ambiziosi tentativi vorrebbero dotare il calcolatore di altre "abilità" umane: riconoscimento della voce, riconoscimento delle "immagini", capacità di tradurre da una lingua all'altra, capacità di eseguire analisi grammaticali e sintattiche, capacità di eseguire "riassunti" di testi qualsivoglia.

Tutti questi tentativi prescindono totalmente dalla comprensione della distanza che esiste tra il pensiero umano e anche il più complicato calcolo finitistico, della varietà enorme di procedimenti mentali (alcuni dei quali analitici e induttivi, non implementabili in modo automatico perché non formali, ma piuttosto basati su "analogie", ad esempio), e infine, del peso enorme che il "contenuto" ha anche nei procedimenti percettivi più elementari. Nella decodificazione del discorso parlato, ad esempio, le ambiguità fonetiche, le imprecisioni di pronuncia, le distorsioni introdotte dal rumore ambientale, vengono corrette e compensate da meccanismi di comprensione del "contenuto" del discorso, dalla consapevolezza del "contesto concettuale" del discorso, che ci porta talvolta a "prevedere" una parola prima che venga pronunciata, o a escludere una interpretazione di un insieme di fonemi in quanto "assurda" in quel determinato contesto. La comprensione del parlato umano, in una parola, *non è un processo puramente percettivo*, ma un mix complesso di percezione e cognizione.

## **Utile, se utilizzato con modestia**

Sia ben chiaro che tutto questo non deve essere inteso come l'affermazione che tutti questi compiti sono in linea di principio, sotto qualunque ipotesi, estranei alle possibilità del calcolo automatico. Essi diventano però possibili solo a prezzo di una fortissima delimitazione e semplificazione del contesto, al punto tale non fare più somigliare queste attività nemmeno lontanamente alle corrispondenti attività umane. Ad esempio, è possibile ottenere il riconoscimento del parlato in una forma anche utile, purché si riducano enormemente il numero di parole pronunciabili, e ci si limiti a costruzioni sintattiche fisse ed elementari,

in modo tale da evitare (artificialmente) determinate pericolose ambiguità fonetiche. Niente a che vedere dunque con la varietà, ampiezza e ricchezza del discorso umano inteso in senso lato, ivi inclusa anche la quotidiana chiacchiera su argomenti banali. Le reti neurali possono avere significative applicazioni per compiti specifici, che però nulla hanno a che vedere con i meccanismi mentali umani (e anche animali). E' forse possibile ottenere un "facitore di riassunti automatico" di testi che non dia risultati ridicoli (come sono invece quelli ottenuti dal facitore di riassunti di Winword97, per esempio). Ma a prezzo comunque di delimitare enormemente, ad un tipo molto ristretto, come contenuto e forma, i testi da analizzare.

In conclusione, ciò che si vuole qui criticare non sono tanto le singole applicazioni, quanto un modo di proporle destinata ad alimentare illusioni e, talvolta, ad ottenere il finanziamento di progetti di ricerca dagli obiettivi infondati.

### **Breve storia di un altro fallimento: Calcolo "predittivo" dei sistemi dinamici e Caos.**

La necessità di mettere in guardia contro pericolose illusioni sulle magnifiche sorti e progressive degli algoritmi, e conseguentemente dei calcolatori e del calcolo automatico non finisce però qui.

Altre dure lezioni sulle limitazione intrinseche dei sistemi formali finiti (anche se giganteschi) ci vengono da altri fronti. Prima di arrivarci, una breve delucidazione su cosa sia un *Sistema dinamico*.

### **Sistemi dinamici.**

Vi sono entità reali che, pur restando perfettamente identificabili rispetto allo sfondo degli altri fenomeni, sono sottoposte ad una loro dinamica interna piuttosto accentuata: si evolvono e mutano forma e aspetto, talvolta molto rapidamente. L'approccio è volutamente astratto, perché le considerazioni che seguono si

applicano ad una enorme e differenziata varietà di fenomeni. Faremo però qualche istruttivo esempio.

Il terreno di coltura di questo secondo genere di "disillusione" è stato, storicamente, la meccanica razionale, e da lì partiremo. Un ramo della fisica è, come noto, la dinamica, intesa come studio del movimento dei corpi a partire dalle sue cause (mentre la cinematica si limita a descrivere compiutamente il movimento, senza spiegarlo). Il problema della dinamica è il seguente: dato un sistema di corpi, essendo nota la configurazione in un certo istante e le leggi di interazione tra i corpi stessi, determinarne l'evoluzione, ovvero le configurazioni che esso assume in tutti gli istanti successivi.

Prendiamo il sistema solare. Esso è composto da un certo numero di masse (il sole, i pianeti, i loro satelliti, gli asteroidi, le comete, ecc.). La legge di interazione è nota, è la legge di gravità. Altrettanto note le leggi generali della dinamica, che legano forza e accelerazione. Se si sviluppano le uguaglianze con le quali si esprimono queste leggi si scopre che si ottengono delle relazioni che legano tra loro grandezze dinamiche che sono delle derivate nel tempo (nel senso del calcolo infinitesimale) una dell'altra. Ad esempio, posizione e accelerazione: l'accelerazione è la derivata - la variazione nel tempo - della velocità. La velocità a sua volta è la derivata - variazione nel tempo - della posizione. Questo tipo di relazioni si chiamano *equazioni differenziali*, dato che sono uguaglianze implicanti appunto funzioni e le loro derivate. Risolvendo le equazioni differenziali (con metodi matematici formali, ove disponibili, oppure "numerici", cioè basati su una approssimazione delle derivate come rapporti tra variazioni finite, ad esempio), si ottengono le posizioni dei pianeti come funzione del tempo (futuro).

La situazione può essere descritta così.



Nel modo summenzionato è possibile "predire" l'evoluzione futura di fenomeni, conoscendo lo stato presente o passato. Come dicevamo, è questo uno schema che può essere adattato a molti fenomeni: oltre al sistema solare, ad esempio, al sistema costituito dall'atmosfera terrestre. In questo ultimo caso la previsione riguarda le condizioni del tempo meteorologico futuro, cosa utile in molte circostanze.

Un sistema formale come sopra descritto viene anche indicato come "modello" del fenomeno (o di un sistema fisico, o reale). I casi in cui la previsione (la risoluzione delle equazioni differenziali) può essere effettuata "con carta e penna" sono ben pochi, e troppo semplici per essere di una qualche utilità pratica. Negli altri casi si ricorre al calcolatore, che viene incaricato attraverso un programma di calcolo, un algoritmo, di "impersonare" il modello, e di eseguire una miriade di calcoli in nostra vece. Non deve essere ignoto, probabilmente, che i modelli di previsione sono stati largamente introdotti anche per i fenomeni macroeconomici (e anche microeconomici).

La disponibilità di calcolatori sempre più grandi e potenti (veloci nell'eseguire gli algoritmi) ha enormemente allargato l'ambito di indagine di fenomeni tradizionalmente considerati insondabili.

Nel corso di queste indagini, storicamente legate alla meccanica e alla termodinamica, sono emersi risultati sorprendenti e in netto contrasto con l'intuizione precedente, ma *di valore del tutto generale*, non legati quindi allo specifico fenomeno o sistema preso in esame.

Per comprendere questi sorprendenti risultati, proviamo a pensare al nostro modello (applicato a quello che volete voi, anche all'evoluzione di una palla da biliardo appena lanciata dal giocatore) come a un qualcosa che, alimentato dalle condizioni iniziali, calcoli (predica) mediante un algoritmo lo stato del sistema ad un istante successivo voluto, che per qualche ragione ci interessa conoscere in anticipo. La previsione sarà ovviamente più o meno giusta, quanto più accurata sarà stata la descrizione formale del sistema (non abbiamo dimenticato variabili importanti, le leggi sono quelle giuste e non ne abbiamo dimenticata nessuna, ecc.) e quanto più corretto è l'algoritmo di calcolo (non abbiamo introdotto semplificazioni eccessive per risparmiare tempo).

Nella figura qui sopra, abbiamo indicato come le equazioni differenziali che reggono l'evoluzione del sistema siano il frutto di applicazioni di leggi "naturali" perfette e deterministiche. Con questo si intende che ci si vuole limitare a considerare sistemi non aleatori, in cui lo stato del sistema sia perfettamente e totalmente determinato dallo stato nell'istante precedente e dall'applicazione di legge certe di evoluzione, senza l'introduzione di nessuna variabile aleatoria, senza nessun "calcolo delle probabilità".

Bene, in questa circostanza ciò che il "senso comune" della meccanica razionale si aspettava è che l'evoluzione del sistema fosse perfettamente determinata dalle condizioni iniziali, per *saecula saeculorum*. Questa era la visione del mondo di grandi pensatori occidentali come Laplace e Cartesio.

Ora fissiamo l'attenzione sulle condizioni iniziali. Esse sono qualcosa che descrive compiutamente ed esaustivamente lo stato di un sistema ad un certo istante, assunto come iniziale.

Se il sistema è un sistema "pensato", cartaceo, possiamo ritenere che i valori delle grandezze che descrivono lo stato iniziale del sistema siano "perfetti", privi di errore. Ma se quello che stiamo studiando è un modello di un sistema reale, in carne ed ossa, dobbiamo forzatamente ritenere che queste condizioni iniziali siano state ottenute attraverso un processo di misura o di stima. Dobbiamo cioè ragionevolmente ammettere che siano soggette ad un errore, per quanto piccolo. Ora, nel luogo comune del pensiero scientifico sette e ottocentesco (ma anche primo-novecentesco), c'era l'idea che i sistemi fossero "buoni", che conservassero cioè questo errore durante il loro evolversi. In altre parole, se si commette diciamo un errore del 10% nella determinazione delle condizioni iniziali, l'aspettativa era che questo errore si conservasse invariato indefinitamente nel corso delle previsioni lontane nel tempo quanto si vuole.

In effetti, esistono sistemi "buoni" siffatti, anzi, tutti i sistemi che si studiavano con "carta e penna" lo erano. Si studiavano infatti solo i sistemi che erano abbastanza semplici da potere essere studiati con "carta e penna", appunto (cosa volete che facesse, d'altronde?). Solo la diffusa disponibilità di calcolatori abbastanza potenti ha permesso di affrontare lo studio degli altri sistemi, dei sistemi complessi non risolvibili appunto con "carta e penna".

Ne è emersa (nel corso diciamo di questi ultimi venti anni) la consapevolezza che questi ultimi non sono affatto "buoni": non conservano cioè l'errore iniziale, ma lo amplificano man mano che la previsione si sposta più in là nel tempo. Più vogliamo prevedere lontano, maggiore diventa l'errore. Spesso questa crescita è, con il tempo di previsione, esponenziale. Questi sistemi mostrano dunque quella che possiamo chiamare una "sensibilità alle condizioni iniziali", che è poi la più corretta definizione di *sistema caotico*.

La circostanza che l'errore, con lo spingersi avanti della predizioni, aumenta indefinitamente ha una conseguenza piuttosto importante e, se volete, sconvolgente, ed è questa: per ogni si-

stema siffatto, dato un errore di stima delle condizioni iniziali piccolo quanto si vuole, esiste un "orizzonte di previsione", un arco di tempo oltre il quale la previsione non ha più senso, perché affetta da un errore grande a piacere (100%, 1000%?). Quanto sia grande questo orizzonte, e come esso dipende dall'errore iniziale, dipende ovviamente dallo specifico sistema, dalle sue caratteristiche "formali" (dalle equazioni differenziali che lo governano) dalla natura della "domanda" che abbiamo in testa della quale cerchiamo una risposta con il nostro modello di previsione. I sistemi nei quali l'errore iniziale cresce esponenzialmente con il tempo di previsione hanno inoltre la sgradevole proprietà di richiedere, per ottenere un allungamento dell'orizzonte di previsione, un proporzionale aumento della precisione nelle condizioni iniziali: per quadruplicare l'orizzonte, bisogna quadruplicare la precisione, e così via.

Qualcuno si potrebbe chiedere, come storicamente è successo, se per caso la crescita così riscontrata dell'errore non dipendesse per caso dal formalismo, dal nostro modo di schematizzare il funzionamento dei sistemi e di calcolarlo. La risposta è no, non è un artefatto del calcolo o del formalismo, ma piuttosto *questo risultato rispecchia l'effettivo comportamento dei sistemi reali*. Siamo dunque autorizzati a parlare, piuttosto che di errore, di vera e propria indeterminazione degli stati futuri, e siamo autorizzati quindi a parlare di comportamento caotico dei sistemi, non solo di "modelli caotici". E tutto ciò nell'ambito di teorie del tutto causalistiche e deterministiche, escludendo a priori la probabilità e, tanto per dire, la meccanica quantistica.

I sistemi veri, i "fenomeni", sono caotici? Sì, praticamente tutti. I sistemi non caotici sono solo quelli libreschi, esistono solo sulla carta.

Esempi? L'atmosfera terrestre è un sistema caotico. Qualcuno ritiene che l'orizzonte massimo di previsione raggiungibile (o meglio, non raggiungibile) per le previsioni meteorologiche si aggiri attorno alla settimana. Il sistema solare è un sistema caotico. L'orizzonte di previsione è particolarmente breve (settima-

ne, giorni) per le orbite molto ellittiche, quali quelle delle comete. Sono caotici i sistemi idrodinamici (la turbolenza, in particolare). E' caotico il semplice pendolo, se solo si fa oscillare il fulcro in alto e in basso a frequenze vicine a quella di oscillazione libera. E' caotico il tavolo da biliardo (anche quello ideale, con le sponde perfettamente riflettenti, e il piano senza difetti). Dopo una decina di rimbalzi, il percorso della biglia comincia ad essere significativamente influenzato dalla forza di gravità espressa dalle persone che stanno attorno al tavolo. Sono infine caotici quei sottosistemi dei nostri sistemi sociali che vanno sotto il nome di "sistemi economici", a qualunque scala li si guardi (molto caotici, a causa delle loro caratteristiche non lineari, e molto mal formalizzati, temo, a causa della difficoltà di descrivere quantitativamente le interazioni con il resto del sistema politico-sociale, cioè *le forze esterne*).

Tutto ciò dovrebbe indurre ad una estrema prudenza nel trarre dalle previsioni fatte mediante modelli matematici conclusioni di carattere particolare o generale, ma non sempre ciò accade. Spesso interviene la "sindrome del calcolatore", sperimentabile anche con le semplici calcolatrici palmari: si digita una operazione e si commette un errore di digitazione. Il risultato è palesemente sbagliato, ma nessuno se ne accorge, perché è "un risultato del calcolatore". Si abdica assieme il senso critico e il pensiero umano.

Il contributo forse maggiore alla sistemazione del pensiero scientifico a proposito dei sistemi caotici ci viene probabilmente dal francese David Rouelle (vivente). Non possiamo però omettere qui un omaggio alla preveggenza di quello straordinario personaggio che fu Henri Poincaré, che ai primi del secolo aveva già più che intravisto tutto ciò (assieme a molte altre cose).

A chi volesse approfondire il tema del Caos e dei Sistemi Caotici, è consigliata la seguente lettura:

David Ruelle, *Caso e Caos* (editore italiano da determinare). Libro a carattere divulgativo (ma anche qui, impegnativo!).

Per chi legge il francese, c'è anche:

A.Dahan Dalmedico, J.-L. Chabert, K.Chemla, *Chaos et déterminisme*, Editions du Seuil, 1992. Si tratta di una raccolta di interventi, sempre a carattere divulgativo, di una quindicina di studiosi che affrontano il tema sotto diverse ottiche (matematiche, fisiche, filosofiche). Non mi risulta una traduzione italiana e sarò grato a chi possa eventualmente segnalarmela.

## **Dell'utilità dei calcolatori, e di come sono fatti.**

Dopo avere saggiamente imparato a diffidare, più che dei calcolatori, dei fallaci risultati che da essi si può ottenere a causa di un loro uso scriteriato, e delle illusioni che sulle loro potenzialità si possono coltivare, è venuto il momento di imparare ad usarli, *perché i calcolatori sono in realtà strumenti molto utili in una infinità di circostanze*, anche se (e forse proprio perché) non pensano e non penseranno mai.

Abbiamo accennato a quasi tutte le problematiche relative ai mezzi di calcolo automatici, adesso è forse venuto il momento di vederle in un'ottica più analitica.

Prima di addentrarci, un breve richiamo: abbiamo già detto che i calcolatori non sono altro che vuote macchine (catoste di interruttori sapientemente collegati tra loro) in grado di eseguire algoritmi. Abbiamo anche inteso come la loro "intelligenza" (la capacità, in senso non umano, di risolvere un determinato problema in modo più o meno efficiente) risieda tutta negli algoritmi, mentre i calcolatori sono solo sede di "forza bruta", di "capacità di esecuzione", più o meno veloci, a seconda di quanto possiamo spendere, e a seconda dell'epoca storica alla quale ci riferiamo.

E' il caso di vedere da vicino qualche algoritmo, e di prendere un po' di pratica con le loro rappresentazioni.

## Algoritmi e programmazione.

Prendiamo due *problemi* piuttosto elementari, ma nei quali credo ognuno si sarà imbattuto almeno una volta nella vita.

**Problema 1:** Data una lista di  $n$  numeri, individuare il maggiore.

**Problema 2:** Data una lista di  $n$  numeri, disporli in ordine crescente.

Il Problema 2 ci risulterà forse più familiare se formulato come segue: Dato un elenco di nomi, disporli in ordine alfabetico. Si tratta di due formulazioni in realtà *dello stesso problema* (il problema dell'ordinamento), come sarà chiaro nel seguito.

Troviamo un algoritmo per il Problema 1. Esprimiamolo per ora a parole, in modo informale, anche se rigoroso. Chiamiamolo Algoritmo 1. Useremo una notazione passo-passo, in questo modo:

Passo:                    Operazione (o Istruzione):

- 
- 1            Scrivere il più piccolo numero possibile nel nostro sistema di rappresentazione (se stiamo parlando di numeri interi, ad esempio, lo zero). Chiamiamo MAX questo numero. (In algebra, scriveremmo:  $MAX = 0$ )
  - 2            Esaminiamo il primo numero della lista ( $a_1$ ). Chiamiamo questo il "numero corrente" (sotto esame). Algebra: corrente =  $a_1$ .
  - 3            Confrontiamo corrente con MAX. Se corrente  $>$  MAX, allora sostituiamo in MAX il valore di corrente. Algebra:  $MAX =$  corrente.
  - 4            Se la lista è finita, andiamo al passo indicato con "FINE". Altrimenti passiamo al successivo numero della lista e chiamiamo questo come corrente. Corrente =  $a_{i+1}$ .
  - 5            Andiamo al passo 3 e proseguiamo di lì.

FINE In MAX è contenuto il massimo (o uno dei massimi) contenuto nella lista.

Semplice ed efficace, come si vede. Qui non è richiesta nessuna particolare "abilità" umana. Basta avere "memoria" (anche artificiale: un foglio dove scrivere i numeri, un "posto" per MAX, dove scrivere e cancellare, un "segno di spunta" da fare sulla lista, per segnare dove si è arrivati, e individuare il successivo). Serve inoltre la capacità di individuare tra due numeri quale sia il maggiore.

Notiamo che al Passo 5 l'istruzione ci dice di tornare indietro, al passo 3. Di lì si va al passo 4 e poi di nuovo al 5, quindi al 3, e così via, finché non finisce la lista di numeri.

Questo percorso circolare prende il nome di "loop" (cerchio). Si continua a girare nel loop, fino a che non si esce per una condizione di fine lista.

Passiamo al Problema 2, e cerchiamo un algoritmo. Semplice!, si potrebbe pensare. Possiamo fare ricorso all'algoritmo 1, che individua il massimo, utilizzandolo più volte. Si potrebbe fare così:

Chiamiamo l'elenco dei numeri disordinati di partenza "lista di ingresso". Chiamiamo "lista di uscita" quella che conterrà i numeri ordinati (all'inizio è vuota)

- 1 Usare l'algoritmo 1 per individuare il massimo nella lista di ingresso.
  - 2 Toglierlo dalla lista dei numeri in ingresso, e metterlo in cima ad un'altra lista, la lista-risultato (la lista di uscita).
  - 3 Se la lista di ingresso non è vuota, andare al passo 1, altrimenti andare al passo FINE.
- FINE La lista di uscita contiene ora i numeri contenuti nella lista di ingresso, ordinati in modo decrescente.

Appendice: Se li si vuole ordinati in modo crescente, copiare la precedente lista di uscita in una nuova lista di uscita in ordine inverso.

Apparentemente il problema è risolto, salvo il fatto che bisognerebbe esprimere queste operazioni di Togliere e Inserire in un modo un po' più formale (richiedente, cioè, meno intuito). Si noti inoltre che abbiamo definito un algoritmo servendoci di un altro. In informatica, un algoritmo scritto in forma riusabile all'interno di un altro prende il nome di routine, o subroutine, e l'uso di un algoritmo siffatto da parte di un altro è detto "chiamata di subroutine". Il passo 1 dell'Algoritmo 2 contiene quindi una "chiamata di subroutine".

Per quanto funzionante, l'Algoritmo 2 non è il colmo dell'efficienza. Si tratta infatti di applicare l'Algoritmo 1 prima su una lista di  $N$  numeri, poi di  $N-1$ , e così via. Quindi l'Algoritmo viene usato  $N+(N-1)+(N-2)+\dots+2+1$  volte. Ovvero  $(N + 1)/2$  volte.

Questo ordine di problemi, i problemi di ordinamento (*sorting* in informatica) è di importanza capitale, e gli algoritmi relativi ricorrono così spesso che si sono fatti enormi sforzi per ottimizzarli. Non ci addenteremo più di tanto, ma un algoritmo migliore (anche se non il migliore) è quello cosiddetto di "bubble sort".

- 1 Si prenda una variabile intera, diciamola  $n$ , e la si ponga uguale a 1.  $n = 1$ .
- 2 Si prenda una variabile binaria o logica, chiamiamola "lavoro finito". La si ponga a "vero". Lavoro finito = vero.
- 3 Si confronti  $a_n$  con  $a_{n+1}$ .
- 4 Se  $a_n > a_{n+1}$ , invertire i due termini e porre Lavoro finito = falso.
- 5 Se  $a_{n+1}$  è l'ultimo della lista, vai al passo n. 8
- 6 Incrementa  $n$  di una unità.  $n \leftarrow n + 1$
- 7 Vai al passo n. 3.

- 8            Se Lavoro finito è vero, vai al passo FINE.  
9            Vai al passo n 1.  
FINE        La lista  $a_n$  è ordinata in ordine crescente.

Nota. Se al passo n. 4 invece di fare il test per  $a_n > a_{n+1}$  si fa il test opposto ( $a_n < a_{n+1}$ ), si ottiene la lista ordinata in ordine decrescente.

Vediamo di capire ora a parole come funziona l'algoritmo. Esso esamina la lista a coppie di elementi adiacenti. Se sono già nell'ordine giusto, li lascia come sono, altrimenti li inverte (li mette cioè nell'ordine giusto). E' facile convincersi che una sola passata in generale non basta, quindi bisogna ripetere il passaggio. Quante volte? Fino a che la lista non risulti tutta in ordine. Come si fa a capire che la lista è tutta in ordine? Semplice: se è tutta in ordine, risulterà che non c'è stato bisogno di effettuare inversioni, e questa è la funzione della variabile logica "Lavoro finito", che viene inizializzata a "vero" ogni volta che si inizia ad esaminare la lista, e posta a "falso" ogni volta che si effettua una inversione.

Questo algoritmo ha la caratteristica che il numero di passi da eseguire per arrivare alla fine del compito non è determinato a priori. Più la lista di partenza è disordinata, più volte è necessario ripercorrerla, e viceversa. Al limite, se la lista è già in ordine, "l'algoritmo l'esaminerà una volta, accorgendosi alla fine che era già in ordine e cessando quindi subito di lavorare" (per usare un periodare antropomorfo).

Come si esprimono, come si scrivono gli algoritmi? Ci sono ovviamente tanti modi, più o meno formali e rigorosi. Quello fin qui usato non è né l'uno né l'altro, anche se ha il vantaggio di richiedere poche precisazioni preliminari per essere compreso.

Limitandosi ai modi tradizionali e tipici dell'informatica, ne esamineremo due: una descrizione in termini di linguaggio di programmazione procedurale, e un diagramma a blocchi.

Linguaggio di programmazione:

<b>Bandierina</b>	<b>Istruzione</b>	<b>Commento</b>
	Program bubblesort	<i>Diamo un nome al programma</i>
	Read LA	<i>Leggi la lunghezza della lista in LA</i>
	Read a(LA)	<i>Copia la lista dei numeri nel vettore a</i>
Inizio_lista:	$N \leftarrow 1$	<i>Poni N uguale a 1</i>
	Lavoro_finito $\leftarrow$ True	<i>Poni Lavoro_finito uguale a vero.</i>
Next_lista:	If $N = LA$ Then goto Finelista	<i>Se si è arrivati alla fine, vai a finelista</i>
	CORRENTE $\leftarrow$ a(N)	
	If CORRENTE < a(N+1) Then $a(N) \leftarrow a(N+1)$ $a(N+1) \leftarrow$ CORRENTE Lavoro_finito $\leftarrow$ False	<i>Se non sono nell'ordine giusto, invertili. Abbiamo invertito, quindi il lavoro non è finito.</i>
	$N \leftarrow N + 1$	<i>Incrementa N (preparati a esaminare il successivo)</i>
	Goto Next_lista	<i>Vai alla relativa bandierina, cioè all'esame delle coppie.</i>
Finelista:	If Lavoro_finito = True Then END	
	Goto Inizio_lista.	
	DATA 5 DATA 0, 7, 5, 11, 1	

Potete provare ad eseguire l'algoritmo con carta e penna. Munitevi di cinque foglietti. Scrivete in alto a ciascuno di essi il nome della variabile che esso ospiterà: Rispettivamente LA, a, N, Lavoro\_finito, CORRENTE.

Seguite le seguenti "Regole numerate":

1. Eseguite le istruzioni dall'inizio, proseguendo alla successiva, salvo quando l'istruzione è un Goto. In tal caso, saltate alla istruzione che porta la bandierina corrispondente.

2. Quando trovate qualcosa del tipo  $X \leftarrow$  espressione, calcolare l'espressione a destra, scrivete il valore nel foglietto X, dopo il valore (se c'è) precedente, e cancellate quest'ultimo con un tratto di penna. Per calcolare l'espressione, servitevi dell'ultimo valore scritto nel foglietto delle variabili corrispondenti. Esempio  $N \leftarrow N + 1$ : prendete il valore di N dal foglietto "N", sommateci uno, scrivetelo dopo l'ultimo valore di N (quello appena usato per sommarci uno), e cancellate quello immediatamente precedente (sempre quello appena usato).
3. L'istruzione "If" significa "Se". Se quel che segue è vero, allora eseguite le istruzioni che si trovano a destra nella stessa riga (separate dai : ) poi eseguite l'istruzione successiva nella riga seguente. Altrimenti saltate alla istruzione seguente senza eseguire la parte a destra.
4. Quando incontrate l'istruzione END, il lavoro è finito.
5. Quando incontrate l'istruzione Read (leggi), cercate la prima istruzione DATA e copiate nel foglietto corrispondente i dati che seguono DATA nell'ordine. Se Read è seguita da una variabile semplice, si legge un dato. Se ciò che segue è qualcosa come  $a(n)$ , allora si richiede di leggere un vettore, qualcosa come  $a_i$ , dotato di n valori. Guardate quanto vale n e leggete dalla DATA tanti valori quanti indicati da n. Ogni volta che leggete un valore DATA, fateci un segno sopra, per non perdere il conto del punto a cui siete arrivati.

Nota finale: siccome a è un vettore con cinque valori ( $a(1)$ ,  $a(2)$  ...  $a(5)$ ), nel foglietto battezzato "a" ricavate una scacchiera di una decina di righe e cinque colonne.

LA	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	N	Lavoro_finito	CORRENTE
5	0	7	5	11	1	1	True	0
	7	7	5	11	1		False	
	7	0	5	11	1	2		0
...	...	...	...	...	...	...	...	...

In questo modo avrete recitato la parte del "calcolatore". Scoprirete che il lavoro è del tutto meccanico: serve diligenza (non si deve mai derogare da nessuna regola) e attenzione (non ci si deve mai distrarre, né sbagliare). In compenso, non è richiesta nessuna iniziativa: dovete seguire le regole (quelle numerate) e le istruzioni del programma. Raggiunta l'agognata END, se avrete fatto tutto come si deve, in capo a un minutino o giù di lì, scoprirete che "automaticamente" la lista degli  $a_i$  è in ordine decrescente (procedendo da sinistra a destra). Scoprirete che lo 0 ha camminato ogni passo verso destra, fermandosi al suo giusto posto all'estrema destra, mentre lo 11 ha camminato verso sinistra, ma questa volta guadagnando una sola posizione ad ogni giro.

Riflettiamo ora solo un attimo su cosa ci è servito per eseguire il compito:

- A. La capacità di sapere scrivere e leggere numeri.
- B. Delle "locazioni di memoria", i nostri foglietti, dotate di un nome di riconoscimento, nel quale memorizzare (scrivere) i valori che la variabile assume man mano che procede il calcolo, "dimenticando" i precedenti (cancellandoli).
- C. La capacità di sapere fare confronti tra numeri, e quindi di stabilire se tra due il primo è minore, oppure uguale, al secondo.
- D. La capacità di applicare meccanicamente le regole numerate (e quindi, di riconoscere le varie istruzioni, sapendo come comportarsi di conseguenza).
- E. La capacità di leggere meccanicamente il programma, seguendo il flusso naturale salvo quando una delle "Regole numerate" non mi dica di fare diversamente.

Queste cinque condizioni sono irrinunciabili, e non ne servono altre, come è facile convincersi. Quindi, qualunque "calcolatore" (sia esso una macchina, o un essere umano ad essa declassato), che voglia eseguire l'algoritmo "bubble-sort", deve essere dotato di queste cinque capacità.

Generalizzando un po' di più, per eseguire algoritmi tecnicamente utili è richiesta la capacità di fare confronti tra i numeri ( = , < , > ) e di fare le quattro operazioni ( + , - , \* , / ). Ma se si accettasse di scomporre queste operazioni in altre ancora più elementari, si potrebbe arrivare ad un insieme di regole veramente minimo, e addirittura ad usare la notazione unaria.

Nota finale: il linguaggio di programmazione qui usato è effettivamente esistente (o almeno somiglia moltissimo ad uno esistente), si chiama Basic, ed è stato rispettato salvo per particolari in questa sede insignificanti.

## **Come sono fatti i Calcolatori**

A questo punto, possiamo farci la faticosa domanda: com'è fatto, e come funziona un calcolatore?

Ovviamente, non è pensabile che esso sia direttamente in grado di leggere un programma scritto nel modo sopraccitato. I calcolatori sono fatti di interruttori (accesi o spenti) e quindi c'è da aspettarsi che il programma debba essere espresso (codificato) in codici binari, rispecchianti lo stato dei famosi interruttori.

In effetti è perfettamente pensabile lavorare in binario: le varie istruzioni del lessico (If-Then, Read, Data, ←, ...) potrebbero essere codificate con un numero binario. I numeri, idem. I nomi delle variabili, anziché con i nomi antropizzanti utilizzati, niente impedisce che siano a loro volta dei numeri. Anche le etichette, non c'è nemmeno bisogno che esistano. Si possono numerare le righe, e riferire i salti al numero di riga, anziché nel modo sopra utilizzato.

Alla fine, in un modo o nell'altro, tutto potrebbe essere ridotto ad una sequenza di numeri binari.

Ma le regole numerate? Nel fare il calcolo a mano, le abbiamo lette, interpretate e "capite" (cioè, applicate alle specifiche situazioni) man mano che serviva.

Ebbene, quelle in effetti è necessario siano in qualche modo "presenti" nei circuiti del calcolatore, esattamente come abbiamo già visto: la capacità di fare le operazioni è contenuta nella forma del circuito (è diverso per somma o moltiplicazione, o per AND e OR). Esse sono, come si dice "cablate" nella macchina, fanno parte della macchina stessa.

## **La CPU (Central Processing Unit)**

Il cuore di un calcolatore è la CPU (Central Processing Unit, Unità centrale di processo). E' lei la grigia e pedissequa - ma velocissima - esecutrice del programma, delle istruzioni codificate in un codice a lei comprensibile che impersonano l'algoritmo che ci interessa. Dentro la CPU sono scritte, nelle interconnessioni tra i vari interruttori che la compongono, le regole di esecuzione delle varie istruzioni, e la regola principe che, in assenza di indicazioni contrarie, finita una istruzione si passa alla successiva. L'operazione di lettura della istruzione (successiva) e l'automatico scatto di tutti i circuiti interni per predisporre a fare la corrispondente operazione, prende il nome di "decodifica dell'istruzione".

Il dizionario di istruzioni è ciò che individua una CPU, ciò che rende diverse le CPU tra loro.

Che tipo di istruzioni? La CPU è munita di registri interni di memoria (gli accumulatori e i registri indice). Negli accumulatori essa copia i dati su cui operare, per poi copiarli a loro volta in memoria. Nei registri indice invece copia gli indirizzi (nel seguito sarà chiaro di cosa esattamente si tratti) dei dati su cui operare. Ogni tipo di CPU ha un suo "lessico" di istruzioni. Ogni istruzione corrisponde ad un codice binario.

Dovrebbe essere chiaro come un programma, per essere eseguibile da una CPU, deve essere scritto utilizzando il suo lessico. Le CPU attuali, almeno le più famose, sono le Intel e com-

patibili, ovvero (oggi) i Pentium. Tutti i Pentium e compatibili adottano lo stesso lessico, e quindi sono in grado di eseguire gli stessi programmi. Altre, con lessici diversi, sono le Motorola 68.000 (dentro i vecchi McIntosh della Apple), i Power PC (dentro i nuovi McIntosh), le CPU Alpha (dentro le Alphastation Digital). Ognuna ha un suo lessico, incompatibile con le altre. Dunque, non stupitevi se un programma scritto per il Pentium non "gira" in un McIntosh (salvo trucchi tremendi che qui non è il caso di affrontare). Vedremo nel seguito che ci sono altre ragioni di incompatibilità, anche tra computer che montano la stessa CPU.

Abbiamo detto "velocissima esecutrice", ma quanto veloce? Le CPU eseguono le istruzioni del proprio lessico in multipli di un tempo fisso, detto "ciclo di clock". Questo tempo è prodotto da un organo, il clock (orologio) che produce questi cicli moltissime volte al secondo. Quante volte? Centinaia di milioni di volte. La velocità del clock, simile ad una frequenza, si misura in Hertz (1 Herz = 1 ciclo al secondo) e suoi multipli. In generale, nella tecnologia dei computer, si usa dell'Hertz una nozione "allargata", che sta per "numero di volte al secondo". Alcune istruzioni richiedono più cicli (sono più complesse) altre meno. In media, però, le CPU contemporanee sono capaci di sfornare quasi una istruzione eseguita a ciclo di clock. Dunque, una CPU Pentium con un clock a 400 MHerz (la più lenta disponibile oggi sul mercato) sforna circa 400 milioni di istruzioni eseguite al secondo. Di unità di misura delle velocità delle CPU ce ne sono molte, ma la più antica e nota è il MIPS (milioni di istruzioni al secondo). Dunque, un Pentium 400 attuale ha quasi 400 MIPS. Esistono altre CPU ancora più veloci. Nella famiglia Pentium esiste l'800 MHerz. In generale, si pensa che entro breve (un anno-due) si raggiungeranno i 1000-1200 Mherzt.

## **La RAM (Random Access Memory)**

Ma i foglietti? Cosa corrisponde ai foglietti di memoria usati per eseguire gli algoritmi? Ad un organo esterno alla CPU, ma a

questa connesso, che si chiama RAM (Random Access Memory). Il nome non tragga in inganno. Non c'è niente di casuale nel suo funzionamento. Essa è semplicemente un organo strutturato a celle (byte, word, a seconda dei casi e delle macchine) nelle quali ogni cella ha un suo "indirizzo", un numero consecutivo. Quando si scrive qualcosa (un numero) in una cella (si agisce sui suoi interruttori), si cancella ciò che precedentemente c'era scritto (ovviamente, muovendo gli interruttori si perde traccia della loro posizione precedente).

Per accedere ad una cella, leggervi quello che contiene o scrivere qualcosa, basta che la CPU comunichi previamente alla RAM l'indirizzo voluto. La CPU può in questo modo saltellare da un indirizzo all'altro senza un ordine prestabilito: questa è l'origine del termine "random access", che va letta quindi come "alla quale si può accedere in maniera casuale". Non tutte le memorie sono necessariamente fatte così. Ad esempio, il nastro magnetico sul quale registrate la vostra musica preferita, non è random, ma piuttosto "sequenziale". Per arrivare al posto n. 1000 (il contatore, ad esempio) dovete svolgere prima tutte le locazioni da 0 a 999, non avete nessun modo (a causa della struttura spiraliforme del nastro avvolto) per "saltarci" direttamente.

Dato che nella RAM ogni cella ha un suo numero-indirizzo, appare abbastanza naturale considerare questo come un "nome di variabile". E' come se, nel nostro programma e nei nostri foglietti, anziché utilizzare dei nomi "umani" (LA, N, ecc.) avessimo chiamato tutte le variabili con un numero: variabile n. 1, n. 2, ecc. Scomodo, per noi, ma in linea di principio non cambia niente.

E il programma (ridotto in forma binaria e corrispondente al lessico della CPU), dove lo mettiamo? Come lo sottoponiamo alla CPU perché l'esegua? Anche quello lo si può mettere nella RAM, istruendo la CPU semplicemente su dove inizia. Esattamente come dovremmo fare con i dati, gli a.

In conclusione, la CPU comunica con la RAM attraverso lo schema indirizzo (posto dove mettere, o leggere) e dato (contenuto dell'indirizzo), entrambi numeri (binari: interruttori che si muovono).

In effetti, avere una CPU e della RAM non basta certo per fare un computer. Cosa potremmo mai farci con un calcolatore nel quale il programma e i dati fossero stati scritti in fabbrica, e i dati-risultati restassero, imperscrutabili, custoditi nella memoria? Servono quindi altri organi in grado di fare dialogare l'insieme CPU-RAM con il mondo esterno: ricevere dati e programmi (in entrambi i casi, sequenze di numeri) e restituire il risultato dell'elaborazione.

### **Organi di ingresso: la tastiera.**

Un buon modo per introdurre dati è la tastiera, ad esempio. Un buon modo per ricevere i risultati dell'elaborazione effettuata, ad esempio, può essere una stampante. Nei primi computer, in effetti, si usavano delle telescriventi, che sono insieme delle tastiere e delle stampanti.

### **Organi di uscita: monitor e stampante.**

Oggi si continua ad usare la tastiera, ma accanto alla stampante si utilizza il Monitor, contenente un tubo a raggi catodici simile a quello contenuto nella vostra televisione. Pur non essendoci nessuna differenza in linea di principio tra un monitor e una stampante (entrambi portano sotto i vostri occhi i risultati calcolati), ci sono molte differenze pratiche: il monitor è molto più veloce e silenzioso, ed è possibile riscrivere su quanto già scritto, rendendo possibile l'animazione (cioè, qualcosa che si muove sullo schermo). In compenso, il monitor è un organo "volatile", come si dice. Se lo spegnete, quel che c'è scritto sopra si cancella. Se in un certo posto dello schermo fate scrivere qualcosa d'altro, le vecchie informazioni spariscono. La stampante è molto più lenta e rumorosa, una volta scritto non può più tornare

indietro e cancellare (non può fare quindi animazioni), ma in compenso produce (quasi) indelebili pezzi di carta, i quali sono per sovrappiù asportabili.

## **Le periferiche e le relative interfacce.**

Stiamo parlando i organi "esterni" al computer addetti all'interazione con l'utente umano, e al computer attaccati attraverso cavi di comunicazione. Questi cavi non possono "parlare" direttamente con la CPU, per una infinità di motivi tecnici, legati essenzialmente alla sproporzione di velocità tra il vostro intervento umano sulla tastiera (o alla lentezza meccanica della stampante) e alle differenti necessità elettriche tra interno e esterno, alla codifica e decodifica dei dati. In una tastiera, ad esempio, serve qualcosa che, se premete il tasto "a", generi il codice ASCII corrispondente. Se premete invece lo stesso tasto, ma assieme a quello di maiuscolo (shift), generi invece il codice ASCII corrispondente alla "A".

Tutti gli organi esterni, quelli cioè delegati a interagire in qualche modo con gli esseri umani, prendono il nome di "**periferiche**". Esse si collegano al computer per mezzo di organi interni chiamati "interfacce di Input-Output" (Ingresso-Uscita) e rivestono, nei calcolatori moderni, una importanza estrema. Il nome "interfaccia" deriva dal fatto che essi hanno in effetti, come Gianno, due facce: una guarda verso l'esterno, verso la periferica, l'altra verso l'interno, il computer propriamente detto, l'insieme CPU+RAM. Il loro compito è di fungere da tramite, in una notevole molteplicità (e varietà) di significati: dal semplice adattamento elettrico, alla conversione di formato, alla memorizzazione temporanea dei dati, allo svolgimento di minuti compiti specifici, elementari e veloci, che sarebbe uno spreco fare eseguire alla CPU.

Anche la RAM, come il tubo a raggi catodici, è "volatile". Se va via la corrente (per ragioni meramente tecnico-costruttive), il suo contenuto si cancella. Dato che noi vogliamo che il nostro calcolatore sia sì un computer (esegua cioè calcoli), ma anche

un *ordinateur*, un gestore e conservatore di informazioni, è opportuno che le informazioni che in qualche modo siamo riusciti ad introdurre non se ne vadano al primo spegnimento. Inoltre, per ragioni tecnico-economiche, il costo per byte della RAM è piuttosto elevato, e essa stessa è piuttosto "ingombrante" (si fa per dire, costo e dimensioni sono in continua evoluzione). Nei computer moderni, la taglia tipica della RAM è dell'ordine delle decine di Megabyte, troppo poco per farne un archivio anche se si trattasse di memoria non volatile.

## **Memorie di massa, dischi.**

Per memorizzare grandi quantità di dati si utilizzano delle unità di memorizzazione "permanenti" completamente diverse, che prendono il nome di "memorie di massa". Esse sono i "dischi magnetici", sia sotto forma di dischi "morbidi" (floppy disk), removibili, che di dischi rigidi, sia removibili che fissi.

Le capacità tipiche dei dischi moderni (informazione valida oggi, e per qualche mese, poi dovrà essere riverificata) sono circa le seguenti:

Floppy tradizionali:	1,44 Mbyte
Floppy "avanzati":	100-250 Mbyte
Dischi rigidi removibili:	250-500 Mbyte.
Dischi rigidi fissi	2-20 Gigabyte.

Abbiamo detto "supporti magnetici": in essi, la scrittura e lettura è fatta in modo analogo ai nastri magnetici, mediante testine e media ferromagnetici. La differenza tra il disco e il nastro è che mentre in quest'ultimo la testina sta ferma e il nastro, avvolto a spirale, gli si svolge sotto (è un supporto sequenziale, come abbiamo già avuto modo di dire), nei dischi il materiale magnetico è spalmato su dei piatti posti in velocissima rotazione (in atmosfera sigillata esente da polvere, nel caso dei dischi rigidi: non vi venga mai in mente di "aprire" un disco rigido!). La testina si muove e va a cercarsi l'informazione sulla superficie. Lo

schema è random, come per la RAM, quindi sempre indirizzato, ove qui l'indirizzo indica le "coordinate" del dato sul piatto. Il costo per byte di un disco rigido è molte volte inferiore a quello della RAM. Viene da domandarsi perché allora utilizzare la RAM, e non direttamente il disco rigido, più capace e meno costoso. La ragione è la velocità di accesso, di diversi ordini di grandezza inferiore nel caso del disco rigido. Quindi, lo schema che si utilizza nei computer (almeno, quelli di oggi) è di conservare l'informazione nel disco rigido (o, più raramente, nei floppy) e di copiarla nella RAM al momento del bisogno. Finito il calcolo, processo contrario: i dati vengono trasferiti dalla RAM al disco rigido, dal quale non si cancellano se non per guasto o per espressa "volontà" (di qualche programma), ma non per semplice spegnimento del computer.

Siccome in tutte le cose è importante la misura, mostro qui di seguito una tabellina con alcune quantificazioni, con l'avvertenza che si tratta di nozioni estremamente deperibili (vita media: qualche mese):

Costo/Kbyte dischi rigidi:	circa 4 Lire.
Costo/Kbyte RAM:	circa 8.000 Lire.
Tempo di accesso RAM:	70 nsec ( $10^{-9}$ sec)
Tempo di accesso Disco:	10 msec ( $10^{-2}$ sec)

*Nota 1: Per tempo di accesso si intende il tempo medio impiegato dalla CPU per raggiungere (in lettura o scrittura) il dato voluto.*

Gli attuali tempi delle RAM corrispondono a frequenze di accesso attorno ai 100 MHz (SDRAM), contro i 400 MHz minimali di velocità della CPU. Gli accessi alla RAM corrispondono quindi oggi ad un "rallentamento" della CPU, un collo di bottiglia. E' però prevedibile, assieme all'aumento della velocità delle CPU, ci sia nel giro di uno-due anni un significativo aumento anche della velocità della RAM.. Sono attese per la fine del '99 memo-

rie da 600 MHz (notare il rapporto 40:1 nel giro di 24 mesi), quindi prossime alle velocità attuali delle CPU.

Oltre ai dischi rigidi, molto diffusi supporti di memorizzazione permanenti sono oggi i CD-ROM (un modo diverso di usare i buoni, vecchi CD per la musica). In essi l'informazione è scritta mediante fori su una sottile pellicola di alluminio (foro = 1, non foro = 0), e sono per questo meno delicati dei corrispondenti supporti magnetici, i quali sono invece sensibili ai campi magnetici (potete cancellare un floppy con una calamita). In compenso sono più lenti dei dischi rigidi, e sono solo leggibili (si possono scrivere in fabbrica, oppure mediante una apposita unità da computer, che però è costosa e lenta).

I CD-ROM attuali hanno capacità di 640 Mbyte utili. La nuova generazione appena uscita si chiama DVD (Digital Video Disk, perché è stata pensata soprattutto per il video), e ha una capacità di 4 Gbyte. Aumenterà nel futuro (prossimo) a 8 e 16 Gbyte.

Questi numeri, sono grandi, grandissimi o stratosferici? Proviamo a confrontarli con qualcosa di familiare. Un libro di 2000 cartelle (un migliaio di pagine: un librone) codificato in ASCII, occuperebbe 3 Mbyte. Quindi questi numeri sembrerebbero enormi. Se però invece di codificare uno scritto vogliamo codificare un'immagine a colori, le cose cambiano. Sullo schermo di un computer le immagini sono composte da puntini luminosi, di diverse sfumature di colore. Anche se non vogliamo una qualità eccelsa servono circa 1 Milione di puntini (pixel) per una pagina a stampa. Associato ad ogni puntino c'è l'informazione di luminosità e colore (due byte, per un risultato medio). Dunque, una schermata a colori contiene già 2 Mbyte di informazione. Se codifichiamo numericamente il suono (come avviene nei CD), 74 minuti di musica stereofonica qualità CD occupano 640 Mbyte.

Un filmatino (anche lui lo si può codificare numericamente) di piccolo formato e della durata di una decina di secondi occupa

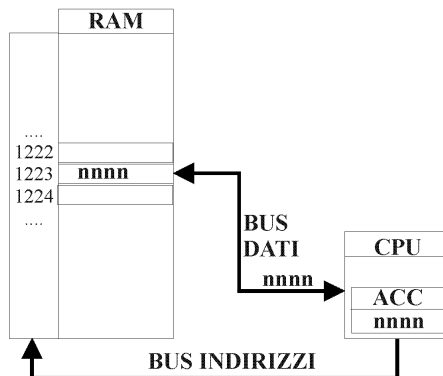
circa 1 Mbyte. Una stampa fotografica a piena qualità occupa diverse decine di Megabyte.

Dunque, se dalla scrittura si passa alle immagini, ai suoni e alla musica (il cosiddetto multimediale), questi numeri cessano di essere grandi. Questa è la ragione della continua rincorsa alle maggiori capacità di memorizzazione, del passaggio dal CD-ROM al DVD, eccetera.

### Schema di un computer. Il flusso interno di informazioni.

Siamo ormai pronti a dare un'occhiata ad uno schema a blocchi di un computer (tipo personal).

Iniziamo dal colloquio CPU RAM. Esso segue lo schema indirizzo-dato, come dicevamo. Per fare questo, esistono due organi di comunicazione distinti, detti rispettivamente *bus indirizzi* e *bus dati*. Sul primo la CPU piazza l'indirizzo del dato che cerca (o che vuole scrivere), mentre sul bus dati viaggiano i contenuti (da o verso la CPU, a seconda se si tratti di lettura o scrittura).



La figura fotografa la CPU sorpresa nell'atto di copiare il numero nnnn dalla memoria di indirizzo 1223 nel suo accumulatore. Come già detto, la CPU è collegata alla RAM con due gruppi di fili: il *bus indirizzi*, con i quali la CPU comunica alla RAM quale indirizzo desidera (per leggervi o scrivervi), e il *bus dati*, con il quale la CPU comunica alla RAM il dato da scrivere (nella locazione stabilita dall'indirizzo presente nel bus indirizzi) oppure la RAM comunica alla CPU il dato richiesto. I dati viaggiano da/verso la memoria verso/da uno degli accumulatori. Gli indirizzi tipicamente viaggiano da qualche registro indice verso il bus indirizzi (quindi, "aprono" la corrispondente cella in memoria)

Lo schema indirizzo-dato è però valido anche per il flusso dei dati tra CPU e tutti gli altri organi interni del computer (non solo la RAM), comprese le interfacce di I/O, le quali tutte si affacciano su questi due bus, e dialogano con la CPU seguendo sempre questo schema.

Cosa distingue la RAM dalle interfacce di I/O? semplicemente il loro set di indirizzi. In un computer, essi sono assegnati in modo che non si sovrappongano mai: da X a Y alla RAM, da A a B alla interfaccia 1, da C a D alla interfaccia 2, e così via.

Come funziona il meccanismo di comunicazione tra tutti questi organi e la CPU?

Lo schema tipico, basilare, di funzionamento è il seguente:

- La CPU legge un dato dalla memoria
- Opera sul dato letto.
- Scrive in memoria il risultato.

La memoria, come abbiamo detto, può essere la RAM, oppure una interfaccia. Grosso modo, mentre leggere e scrivere dati in memoria rappresenta un momento di un calcolo (e quindi, lo schematico processo qui descritto rappresenta l'accumulazione di un risultato intermedio o parziale), leggere da una interfaccia significa prelevare un dato dall'esterno (es., il codice ASCII del

tasto che avete appena premuto), e scrivere su una interfaccia significa "emettere" un dato verso l'esterno (esempio, la lettera "a" che vi trovate scritta sul monitor come risultato di una operazione X).

Se si desidera capire con appena un po' più di dettaglio lo schema di cui sopra, ci si può rappresentare il meccanismo nel modo descritto dall'esempio che segue:

1. La CPU legge un dato dalla memoria e lo interpreta come una istruzione.
2. La istruzione è di caricamento dalla memoria in un registro. Il dato seguente nella memoria indica in genere l'indirizzo nel quale trovare il dato da caricare.
3. La CPU carica il dato successivo (che è l'indirizzo del dato su cui si deve operare) e lo pone in un registro indice, e quindi sul bus indirizzi.
4. La CPU legge il dato su cui operare nella cella di memoria "aperta" dall'indirizzo presente nel bus indirizzi. Lo copia cioè in uno dei suoi registri (accumulatore o registro indice, a seconda dei casi).
5. La CPU opera sul dato appena caricato. Il risultato è sempre in uno dei suoi registri interni.
6. La CPU legge un dato dalla memoria e lo interpreta come una istruzione.
7. La istruzione è di scrittura in memoria del dato (risultato della operazione) contenuto in un registro. Il dato seguente (nella memoria) indica in genere l'indirizzo nel quale scrivere il dato.
8. La CPU carica il dato successivo (che è l'indirizzo dove deve essere scritto il risultato dell'operazione) e lo pone in un registro indice, e quindi sul bus indirizzi.
9. La CPU scrive il risultato nella cella di memoria "aperta" dall'indirizzo presente nel bus indirizzi. Lo copia cioè da uno dei suoi registri (accumulatore o registro indice, a seconda dei casi).

10. La CPU legge dalla memoria il dato successivo e lo interpreta come una istruzione. Il ciclo ricomincia.

Se le istruzioni descritte si riferiscono a operazioni di lettura/scrittura su RAM, è facile notare come questa contenga successioni di numeri che sono interpretabili (e dalla CPU interpretati) sia come codici di istruzione, che come indirizzi, che come dati. Inoltre, di norma, l'esplorazione della RAM, almeno per quanto riguarda i codici di operazione e i relativi indirizzi, avviene in modo sequenziale, una locazione dopo l'altra. Il tutto è abbastanza simile alla "esplorazione" degli algoritmi fatta "a mano" nel capitolo relativo. Lì però avevamo previsto la possibilità di interrompere il flusso regolare (una riga dopo l'altra) mediante istruzioni di "salto" (GoTo) a righe etichettate. Anche la nostra CPU, in tutta analogia, deve essere capace di eseguire salti di questo genere, interrompendo il regolare flusso del programma, sia come risultato di un confronto tra numeri (ricordate? If  $A < B$  GoTo ...), sia in modo incondizionato (come nel caso dei loop).

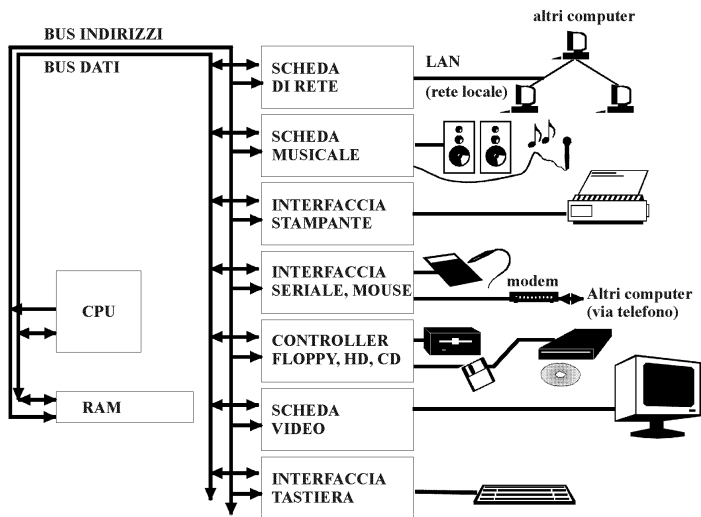
Questo vuol dire che nel "lessico" della CPU esisteranno codici di operazione che indicano la lettura, la scrittura, operazioni varie di tipo aritmetico sui contenuti dei registri interni, l'esecuzione di test logico-aritmetici ( $A1 > A2?$ ), e infine l'esecuzione di "salti" ad indirizzi diversi da quello successivo.

Stendere un programma per una determinata CPU (cioè, scrivere un algoritmo per risolvere un determinato problema nei termini del lessico di quella CPU) significa stendere un elenco di codici-indirizzi-dati che, sottoposti alla CPU, producano il risultato voluto. Quando si ritenga di dovere introdurre un dato dall'esterno (rispetto alla coppia CPU-RAM), occorrerà introdurre una opportuna sequenza che provveda alla lettura di questo dato (ad esempio, il tasto premuto) dalla relativa interfaccia, quindi una *lettura dagli indirizzi occupati da quella interfaccia*. Quando si ritenga di avere raggiunto un risultato da comunicare all'operatore (o semplicemente, da inviare da qualche parte fuo-

ri del set CPU-RAM), occorrerà introdurre una sequenza che scriva i relativi dati-risultato sulla interfaccia che ci interessa. Facciamo notare come anche il disco rigido, o il floppy, sia qualche cosa di "esterno" alla coppia CPU-RAM. Quindi anche le operazioni di lettura e scrittura di dati sul disco sono operazioni di Input-Output, per le quali si deve passare per una opportuna interfaccia.

## Le interfacce.

Ma quali sono queste interfacce? In teoria, non c'è limite. E' infatti sempre possibile inventarsi una nuova periferica. Riferendoci ad un uso tipico odierno, però, è possibile affermare che queste sono le principali e più comuni.



Qualche commento: la figura rende evidente il ruolo di "Giano bifronte" svolto dalle interfacce (o schede). Alcune prendono

nomi più specifici, come il "controller" per i floppy, i dischi rigidi (Hard Disk), e i CD.

Esaminiamone però una alla volta, iniziando dall'alto.

**Scheda di rete.** Essa serve per collegare il computer alla Rete Locale (Local Area Network, LAN), ovvero in buona sostanza un cavo elettrico di determinate caratteristiche che permette di collegare diversi computer tra di loro. In questo modo è possibile scambiarsi o condividere dati e informazioni con altri utenti senza muoversi dal proprio posto. Torneremo più in là sulle reti locali.

**La scheda musicale** dà "voce" e "suono" al computer. Essa permette di ascoltare suoni prodotti dal computer, di registrare (numericamente) dei suoni provenienti dall'ambiente esterno (attraverso il microfono, ad esempio). Permette infine di fare suonare strumenti musicali elettronici MIDI, e di utilizzare una comoda interfaccia per videogiochi: il famoso "joystick", una levetta che potete muovere in alto, in basso, a destra e a sinistra per pilotare automobili, aerei, elicotteri, ecc. ecc. virtuali.

**L'interfaccia stampante** (o porta parallela, o porta Centronics) permette di collegare, appunto, una stampante. Sulle stampanti moderne è possibile stampare oltre ai testi, disegni e fotografie (a colori o in bianco e nero, a secondo della stampante).

**Interfaccia seriale/mouse.** Dette anche porta mouse (se il mouse è tipo PS/2), e porte RS232. A queste ultime potete collegare ad esempio o un mouse seriale (non PS2), oppure un MODEM esterno (MODulator-DEModulator, Modulatore-Demodulatore). Questo dispositivo permette di collegare tra loro computer "lontani" attraverso la normale linea telefonica, oppure di mandare/ricevere FAX. Se non si possiede un collegamento permanente (come è invece il caso delle Università), esso è lo strumento principe per collegarsi ad Internet, la rete mondiale di calcolatori.

**Controller floppy/HD/CD.** Ad esso di collegano i "drive" (unità di pilotaggio), che sono i componenti fisici nei quali alloggiavano fisicamente i floppy, o i CD. Il disco rigido normalmente è inter-

no al computer, e da fuori non si vede, ma anche esso è collegato al Controller (o, è più propriamente, "Host Adapter").

**Scheda Video.** Genera, su comando della CPU, i segnali video necessari a pilotare il monitor, e quindi consente alla CPU di "scrivere" o "disegnare" sul vostro video. Le caratteristiche delle schede video (non sono tutte uguali) sono:

1. **Le risoluzioni supportate**, ovvero il numero di puntini luminosi (pixel) che sono in grado di piazzare sullo schermo. Più sono numerosi, più l'immagine è definita. Risoluzioni standard sono (orizzontale x verticale):
  - 640 x 480: VGA (Video Graphic Array) classica. 307.200 pixel in tutto.
  - 800 x 600: Super VGA. 480.000 pixel.
  - 1024 x 768: 786.432 pixel
  - 1280 x 1024. 1.310.720 pixel.
2. **Il numero di colori supportati**, alle varie risoluzioni. I valori standard sono:
  - 16 colori: VGA classica, delle origini, oramai del tutto insufficiente. Un Nibble/pixel.
  - 256 colori: Super VGA. Nelle fotografie si intravedono "salti" nelle sfumature. Un byte/pixel.
  - 65536 colori: Oramai il minimo indispensabile. Le fotografie danno un buon risultato. Due Byte/pixel.
  - True Color: 16.777.216 colori. Per usi professionali, ma oramai usuale (Tre byte/pixel).
3. **La velocità di scrittura sul monitor**, ovvero al velocità impiegata dalla scheda a "disegnare" sul monitor quanto richiesto dalla CPU. Inutile avere un computer veloce se la scheda è lenta. La velocità si misura in Megapixel/secondo, ma occorre tenere conto anche di altri fattori: alcune schede (le cosiddette "accelerate") gestiscono loro il 3D (tre dimensioni, ovvero la generazione della prospettiva) e i riempimenti di colore di poligoni, e raggiungono maggior velocità. Altre, più semplici, delegano questi compiti al calcolo

della CPU, e a causa di ciò la formazione delle immagini sullo schermo è più lenta.

4. **Interfaccia tastiera.** Si incarica di generare, alla pressione dei vari tasti, i relativi codici (e di accendere/spegnere le varie spie).

## Organizzazione dei dati nelle memorie di massa

Da quanto detto si sarà capito che il calcolatore, nella sua accezione di “ordinateur”, cioè di conservatore e gestore di informazioni, deve molto alla sua memoria di massa, al disco rigido. Ma come è organizzata l'informazione sul disco? I criteri e le regole di organizzazione devono essere, dato che dovranno essere gestiti da algoritmi, da una parte assolutamente rigidi e predeterminati, senza ambiguità, ma dall'altra dovranno permettere facilmente di ritrovare le informazioni che vi sono state memorizzate, di cancellarle, di modificarle o aggiungerne delle nuove.

Il compito di trovare un buon modo di organizzare tutto questo non è, a prima vista, facile. E in effetti, è così. Ci sono molti modi di concepire l'organizzazione dell'informazione su un disco, molti ce ne sono stati e molti ancora se ne potranno inventare in futuro. Qui ci limitiamo alle cose fondamentali, e alla organizzazione tipica del Sistema Operativo (più oltre sarà chiaro il significato di questo termine) attualmente più diffuso: il DOS (Disk Operating System) della Microsoft, che è alla base anche di Windows 3.11 e Windows 95.

Per spiegarci, faremo uso di due approcci opposti, ma convergenti: Il primo, dall'alto in basso (Top-Down), partirà dall'organizzazione così come la vede l'utente umano, e dalle esigenze di quest'ultimo. Il secondo, dal basso verso l'alto (Bottom-Up), partirà invece dall'organizzazione dei singoli byte sul disco, e dalla sua struttura fisica. La trattazione di questo se-

condo approccio sarà sviluppata in stretta correlazione con l'illustrazione delle funzioni del Sistema Operativo.

Iniziamo quindi da ciò che "vede" l'utente.

L'unità di informazione fondamentale è il File. Questo termine potrebbe essere tradotto con "Archivio", ma potrebbe dare luogo ad equivoci. Si preferisce quindi utilizzare direttamente il termine inglese. Cos'è un File? Esso è una sequenza ordinata di byte contenente la codifica di una unità di informazione intesa nel senso dell'utente. Un File potrebbe contenere, ad esempio, una lettera scritta ad un vostro corrispondente (codificata in ASCII), oppure una lista di numeri da ordinare con uno degli algoritmi di ordinamento esaminati. Può contenere anche un programma, ovvero un algoritmo codificato secondo il lessico e la sintassi della CPU montata dentro il vostro computer.

Per potere distinguere un file dall'altro, è possibile assegnare ad esso un nome (*filename*). Un nome di file, nei sistemi operativi moderni, è una stringa di caratteri, composta secondo una certa sintassi. Il DOS, ad esempio, permette di assegnare come nome una stringa composta di due parti: il prefisso e il suffisso (detto anche "estensione" o "extension" del File). Il prefisso può essere composto da un massimo di otto caratteri alfanumerici (cioè, lettere dell'alfabeto o cifre decimali) e il suffisso di tre.

Dato che nei dischi l'informazione è organizzata per file, il modo scelto di organizzare i file prende il nome di *File System*. Ce n'è diversi: quello DOS, quello della ultima versione di Windows95, quello del McIntosh, quello specifico di WindowsNT (che ha il nome NTFS, New Technology File System), quello di McOS (Apple McIntosh), quello di Unix, e così via. Ognuno ha le sue particolarità.

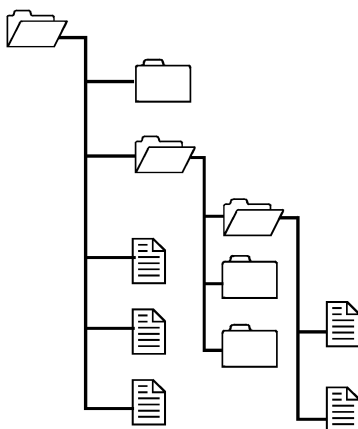
Detto questo, si somigliano però molto, anche nell'affrontare il problema che segue. Quando si ha a che fare con i moderni dischi rigidi, con capacità enormi, organizzare l'informazione per file non è sufficiente. I file possono essere così numerosi da rendere difficile ritrovarne uno in mezzo a tanti, e ancora più difficile evitare le omonimie (cioè, desiderare di chiamare due file




con lo stesso nome). Le omonimie sono ovviamente vietate: dato che l'unica cosa che distingue un file dall'altro è il suo nome, chiamare due file con lo stesso porterebbe ad una ambiguità insolubile.

Per ovviare a questi inconvenienti si è introdotta un'altra unità informativa: l'indirizzario (o folder, o directory, o cartella). Anche esso dotato di nome, non è però che un "contenitore": di altre cartelle o di file. Per usare la metafora tipica degli archivi da ufficio, la cartella sta al faldone come il file sta al documento. Solo che negli archivi cartacei non si va molto oltre un paio di livelli. Se aprite un faldone (un "contenitore") potrete trovarvi dentro o documenti (lettere, ad esempio) o cartelline (altri "contenitori", entità cioè omogenee al faldone). Dentro le cartelline normalmente trovate solo documenti (e non altre cartelline che contengono altre cartelline, ecc.).

Nell'organizzazione dei dischi (*File System*) invece il gioco delle scatole cinesi può andare avanti all'infinito: dentro una cartellina potrete trovare enne altre cartelline e qualche file. Dentro ciascuna delle cartelline altre cartelline ancora e qualche file, e così via all'infinito.

In questo modo è possibile strutturare fortemente l'informazione secondo un principio strettamente gerarchico, o, se volete, ad albero. Un buon modo di rappresentare questo tipo di gerarchia è infatti quello dell'albero (stilizzato). La circostanza che una cosa (cartellina o file) sia contenuta in un'altra (cartellina) è indicata con un tratto, in questo modo.



Qui vediamo delle cartelle, indicate con i simboli  e  (alcune chiuse, altre aperte, mostranti, cioè, il loro contenuto) e dei file, indicati con il simbolo , contenuti nelle cartelle (aperte: di quelle chiuse vediamo l'esistenza, ma ne ignoriamo il contenuto). Cartelle e file possiedono normalmente un nome, assegnato dall'utente, per rammentargli il contenuto (es: cartella "lettere spedite", o cartella "conti della spesa").

## Sistema operativo: DOS e Windows 3.1

### DOS Microsoft

Il sistema operativo più diffuso dei personal computer compatibili IBM è, ancora oggi, il DOS. Per quanto possa non riscuotere i nostri entusiasmi, dobbiamo impararne almeno i rudimenti.

Esso appartiene a quei sistemi operativi con interfaccia "a carattere", intendendo con questo che il loro modo di gestire il dialogo con l'utente è attraverso comandi composti da stringhe digitate sulla tastiera, e attraverso messaggi scritti, stampati sullo

schermo una riga dopo l'altra come si farebbe su di una tele-scrittore. L'interfaccia a carattere esclude quindi l'uso di immagini o disegni nell'interazione tra utente e Sistema Operativo.

Se accendete un computer munito del DOS, dopo una fase iniziale durante la quale il DOS viene appunto caricato dal disco rigido (copiato, cioè, dal disco rigido nella memoria RAM), noterete che sul video compare una piccola scritta:

```
C:\>
```

Questo è quello che si chiama il "prompt" del sistema operativo, e in questa esatta forma è il prompt del DOS. Esso vi indica che il Sistema Operativo si è impadronito del vostro computer ed è pronto a ricevere i vostri comandi digitati sulla tastiera.

## *File system DOS*

Cosa indica il prompt del DOS?

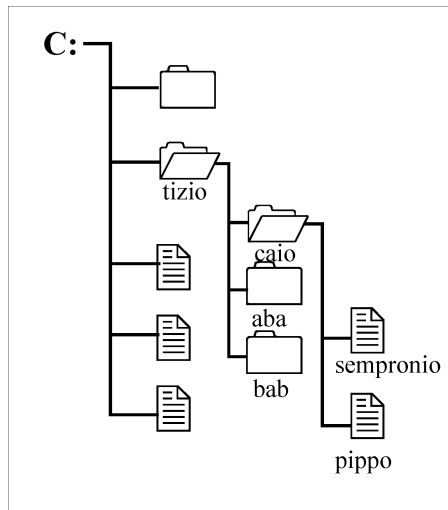
Il file system del DOS è del tutto simile a quello che è stato descritto nel capitolo precedente: nel disco, l'informazione è raccolta in cartelle (directory, in linguaggio DOS, o indirizzari), le quali possono contenere o file, o a loro volta altre cartelle, e così via.

In cima (anzi, alla radice) dell'albero c'è il disco rigido. In DOS, i dischi rigidi si indicano convenzionalmente con una lettera seguita dai due punti. Il disco A è il floppy, mentre la lettera B è riservata all'eventuale floppy aggiuntivo. C è la prima lettera riservata al primo disco rigido (quello dove è memorizzato il sistema operativo, e dal quale viene quindi caricato. Il disco che contiene il sistema operativo prende il nome di "disco di boot". Altri dischi rigidi, o CD-ROM, occuperanno le lettere successive. Dalla struttura ad albero del file system, è facile capire come, a partire dalla radice, per raggiungere un file (e quindi, per indicarlo in modo univoco in tutto il disco), c'è da fare un "percorso", come in un labirinto. Questo percorso parte dal disco rigido in questione, e scende per tappe successive attraverso le cartelle fino a raggiungere il file. Dato che le cartelle hanno tutte un

nome, per indicare dove si trova il file (e quindi, di quale file si tratti), basta indicare questo percorso (in inglese: path). Ciò lo si fa seguendo una semplice sintassi. A partire da sinistra, si scrive prima il disco in questione con la sua lettera seguita dai due punti. Quindi si inserisce un “separatore”, un carattere che non può essere utilizzato per comporre i nomi delle cartelle e dei file, e che quindi può senza ambiguità indicare dove finisce il nome di una cartella e dove inizia quello di un'altra.

Dopo il separatore, si mette il nome della cartella più in alto nella gerarchia, quindi un altro separatore, quindi il nome della cartella immediatamente successiva, quindi un altro separatore, e così via fino al nome del file. Come carattere separatore si usa la barra rovesciata “\”. **Attenti a non confonderla con la barra dritta “/”**, che si trova normalmente invece sopra il “7”.

Esempio:



Il cammino per raggiungere il file “sempronio”, secondo la sintassi DOS, è:

```
C:\tizio\caio\sempronio.
```

Questa stringa individua univocamente il file. Se ci fosse un altro "sempronio" nel disco, ma in un'altra posizione, la stringa sarebbe diversa. Il DOS ammette dunque file con lo stesso nome, ma non con lo stesso cammino. Quindi, vietate cartelle e file nello stesso "posto" con lo stesso nome.

Altra caratteristica del DOS è quella di non distinguere le maiuscole dalle minuscole. Quindi:

```
c:\tizio\caio\sempronio
```

```
C:\TIZIO\CAIO\SEMPRONIO
```

```
C:\Tizio\Caio\Sempronio
```

Indicano tutte la stessa cosa.

### **I nomi delle cartelle e dei file**

Il DOS permette di chiamare sia le cartelle, che i file, con dei nomi composti di caratteri alfanumerici, ma impone certe limitazioni. Le regole sono le stesse per file e cartelle.

Ogni nome è composto da due parti: un prefisso e un suffisso (detto anche "estensione" o "extension", in inglese): un po' come per le persone si usa un nome e un cognome.

Il prefisso può essere composto da un massimo di otto caratteri.

Il suffisso da un massimo di tre.

Sono vietati i seguenti caratteri: \ \* ? / . " così come lo spazio bianco.

Quando si indica il nome di un file, si divide il prefisso dal suffisso con un carattere punto ".".

Dunque:

```
12.2
```

```
Mio.fil
```

100.bat

mio.001

Sono tutti nomi di file validi.

m.i.000

m\o.1

m?o.000

sono tutti nomi non validi. Ma non vi preoccupate, se vi sbagliate, il DOS vi avvertirà e non vi permetterà di creare file con nomi non validi.

### **Cartella attiva, ovvero il “luogo del disco dove ci si trova”**

Il DOS (e con lui molti sistemi operativi) permette di aprire una sola cartella alla volta. Si dice in questo caso che quella cartella è “aperta”, o “attiva” o che “ci si trova dentro di essa”.

Appena acceso il computer, la cartella attiva è la radice del disco di boot (Normalmente, C:\). Per rendere attiva un'altra cartella (o, se volete, per “andarci”) si devono utilizzare alcuni comandi del DOS, come vedremo del capitolo successivo.

In ogni caso, il DOS vi ricorda “dove state” mostrandovi il cammino della cartella attiva dentro il prompt, che è composto da questo cammino seguito dal segno “>”.

Ecco spiegato il senso del prompt alla partenza:

```
C:\>
```

Se si andasse nella cartella dove si trova “sempronio”, otterremmo il prompt:

```
C:\TIZIO\CAIO>
```

La cosa ha la sua importanza (e utilità), perché quasi tutti i comandi DOS operano, se non specificato diversamente, sulla

cartella attiva. E' bene quindi non perdere mai d'occhio dove ci si trova.

### *I comandi principali*

Per impartire al DOS un comando, basta digitarne il nome, subito dopo il prompt, senza spazi bianchi, e premere il tasto "Invio" (↵) per segnalare che il nome del comando è finito.

Ma come si fa a sapere quali sono i comandi disponibili? Semplice, bisogna ricordarseli a memoria, oppure consultare un manuale, oppure digitare il comando "help" (aiuto).

Per fortuna, non sono tanti, e quelli principali ancora meno.

Vediamoli uno per uno.

#### **Cambiare disco**

Come si fa a cambiare disco? (Sono in C, ma voglio andare in A).

Semplice: Battere come comando il nome del disco, seguito dai due punti, poi "Invio".

A:↵

Ad esempio, "vi porta" nel floppy. Subito dopo otterreste il prompt:

A: >

#### **Cambiare cartella**

Una volta scelto il disco, come si fa ad andare in una cartella (a renderla attiva)?

Si usa il comando CD (Change Directory), seguito da uno spazio, e dal cammino per raggiungere la cartella a partire dal punto dove già si stà. Esempio, se si vuole raggiungere la cartella dove sta il file "sempronio", e si sta in A:, bisogna digitare:

C: ↵ (per andare in C)

C:\> (Prompt del DOS che indica che si è andati in C).

C:\>**CD TIZIO\CAIO** (cambia directory, e il cammino a partire da C:\)

Attenzione! Nella riga precedente il comando da digitare è solo quello in grassetto. Il prompt a sinistra, in carattere normale, viene scritto dal DOS! Nel seguito useremo sempre questo modo di scrivere, perché riproduce quello che si vede scritto sul video, visto che i comandi vengono digitati sulla stessa riga del prompt, subito dopo. Il DOS non mette il grassetto, però, siete voi che dovete distinguere il prompt (la stringa più a sinistra che finisce con il > compreso) dal resto digitato da voi.

Posso anche fare un gradino alla volta:

C:\>CD TIZIO

C:\TIZIO>CD CAIO

C:\TIZIO\CAIO>

Infine, esiste un modo per riferirsi alla cartella in cui ci si trova (indipendentemente da come si chiami) e a quella immediatamente superiore (indipendentemente da come si chiami).

Si tratta di due “nomi” particolari:

. = Questa cartella.

.. = La cartella superiore.

Può sembrare strano che esista un modo per indicare “qui”, “questa cartella”, ma si tratta in realtà di una cosa indispensabile, soprattutto dal punto di vista della programmazione. “.” Indica la cartella attiva, indipendentemente dal suo nome.

C:\TIZIO\CAIO>CD ..

C:\TIZIO>

Ancora, posso “saltare” direttamente alla radice, che si indica convenzionalmente con “\” (come se il primo separatore nel cammino completo indicasse “radice del disco in cui ci si trova”):

```
C:\TIZIO\CAIO>CD\
```

```
C:\>
```

### **Elencare i file e le cartelle contenuti nella cartella attiva**

Una volta arrivati in una qualche cartella, sarà bene avere a disposizione un comando per vedere cosa c'è dentro (altrimenti dovremmo ricordarcelo a memoria, cosa non facile quando si hanno migliaia di file).

Questo lo si fa con il comando DIR (Directory), seguito eventualmente dall'opzione “/P”, per mostrare una pagina alla volta, aspettando la pressione di un tasto per andare alla pagina successiva (cosa utile, quando cartelle e file sono più di venticinque, quante, cioè, le righe che può mostrare contemporaneamente lo schermo).

Esempio, ci troviamo in TIZIO:

```
C:\TIZIO> DIR
```

```
CAIO          <DIR>                02/02/98   1.09
ABA           <DIR>                20/01/98  16.08
BAB           <DIR>                20/01/98  16.08
              0 file          0 byte
              3 dir           328.564.736 byte di-
              sponibili
```

```
C:\TIZIO>
```

Se ci troviamo invece in CAIO:

```
C:\TIZIO\CAIO>DIR
```

```
SEMPRONIO          271   19/01/98   3.58
PIPPO              1     23/02/98  21.44
                   2 file      272 byte
                   0 dir       328.564.736 byte disponibili
```

```
C:\TIZIO\CAIO>
```

Qual è il significato dell'elenco fornitoci a video dal DOS?

- Ogni riga elenca una cartella o un file.
- Se si tratta di una cartella, il nome è seguito dalla stringa <DIR>, seguita a suo volta dalla data e dall'ora nella quale la cartella è stata creata.
- Se si tratta di un file, il nome è seguito dalla sua lunghezza in byte, seguito dalla data e ora di creazione del file.
- Seguono due righe di informazioni sintetiche: il numero di file elencati, seguiti dall'occupazione totale, e il numero di cartelle elencate, seguite dal numero di byte ancora disponibili sul disco.

Se la mia cartella è molto affollata, e voglio solo sincerarmi che un file esiste, oppure controllarne la dimensione, e il tempo di creazione, posso fare seguire a DIR uno spazio seguito dal nome del mio file o cartella, e il DOS mi farà vedere solo la riga relativa.

La cosa potrebbe sembrare di poco conto, se non fosse per il fatto che esiste in DOS un modo per riferirsi ai nomi di file in modo collettivo, o sintetico, se volete, sfruttando aspetti comuni del loro nome.

Questo si fa utilizzando, nello specificare il nome del file, due caratteri "jolly", che indicano rispettivamente:

? = qualunque carattere

\* = qualunque stringa di caratteri.

Dunque, per ottenere l'elenco dei file che si chiamano "qualcosa".DOC (documenti, presumibilmente), basta digitare: DIR \*.DOC.

### **Cancellare file**

Come si fa a cancellare un file indesiderato (o non più utile?). Usando il comando DEL (DELeTe, distruggi), seguito dopo uno o più spazi bianchi dal nome del file da cancellare. Naturalmente, ci si deve già trovare nella cartella dove il file si trova. Attenzione, se dopo avere cancellato il file, vi pentite, non c'è modo di recuperarlo se non sono stati predisposti dei sistemi di "Cestino" (non inclusi nel DOS!).

```
C:\TIZIO\CAIO>DEL SEMPRONIO
```

```
C:\TIZIO\CAIO>
```

C'è un aspetto terrificante del comando DEL che è opportuno conoscere, ed è il seguente. Se dovete cancellare centinaia di file, è un po' scomodo dovere digitare una riga per file, ciascuna con il nome appropriato. Supponiamo di avere in una cartella un centinaio di file che si chiamano tutti "qualcosa".TMP (nome tipico di file temporanei, generalmente inutili una volta usati). Per riferirli collettivamente, posso fare uso dei caratteri "jolly". Se li voglio cancellare tutti mi basterà dunque digitare:

```
C:\TIZIO\CAIO>DEL *.TMP
```

Ovvero: cancellare tutti i file che si chiamano "qualunque cosa".TMP.

Supponiamo invece di avere dei file che si chiamano MIO.001, MIO.002 ... MIO.00Z, e di volerli cancellare tutti con un sol colpo: Basta digitare:

```
C:\TIZIO\CAIO>DEL MIO.00?←
```

Cioè, tutti i file che si chiamano MIO.00"qualunque carattere". Dov'è l'aspetto terrificante? Sta nella circostanza che, se tra le centinaia (o migliaia) di file che in questo modo in un sol colpo si possono fare sparire, ce n'è uno che rispetta i criteri indicati, ma che non volevamo cancellare, esso sparisce senza un fiato assieme a tutti gli altri. Esempio: cancello allegramente tutti gli A\*.\*, perché sono infestato da molti file che si chiamano A00010.000, A100200.032, ... e non mi avvedo che c'è un file che si chiama AMATA.MIA, contenente appunto una importante lettera alla suddetta, la quale viene così distrutta per sempre assieme a tutti gli altri.

L'aspetto ancora più terrificante sta nel fatto che posso (e a volte, devo) tranquillamente digitare DEL \*.\* , (cioè: cancella **tutti** i file contenuti nella cartella), e il DOS eseguirà in un lampo, dopo avermi semplicemente avvertito:

Tutti i file nella directory verranno cancellati.

Proseguire con l'operazione (S/N)?

Se premo il tasto "S" è fatta, tutta la directory sarà svuotata dei suoi file senza rimedio.

Attenti quindi a **dove** si mettono i file: separateli in cartelle diverse, con criteri razionali, non ammucciate cose diverse nella stessa cartella! Non usate cartelle dove conservate file importanti e omogenei per depositarvi file temporanei e come "cartella di passaggio"! Create, piuttosto, delle cartelle apposite.

### **Creare una cartella**

Come si fa a creare una nuova cartella? Ci si mette là dove si la vuole creare, e si usa il comando MD (Make Directory), seguito da uno o più spazi bianchi e dal nome che si vuole attribuire alla nuova cartella (che le resterà indissolubilmente legato, salvo non decisi di cambiarle nome).

```
C:\TIZIO\CAIO>MD CARTELLIN←
```

Attenzione, se cercate di creare una cartella con un nome di una cartella già esistente là dove tentate di crearla, l'operazione non riesce e il DOS vi stampa un messaggio di avvertimento:

La directory esiste già (se il DOS è in italiano)

In questo modo non potrete creare neanche per errore situazioni ambigue.

### **Eliminare una cartella**

Ovviamente, potete eliminare una cartella, purché sia però vuota. Per svuotarla, semplice: basta andarci dentro e fare DEL \*.\*! Peggio ancora, potete dalla cartella immediatamente superiore fare DEL CARTELLIN. Perché peggio? Perché è buona regola, prima di cancellare tutto, dare una occhiata a quello che c'è dentro. Fate quindi un DIR, o anche un DIR CARTELLIN, stando sempre nella cartella immediatamente superiore.

Una volta svuotata, una cartella può essere eliminata con il comando RD (Remove Directory), seguito da uno o più spazi, e dal nome della cartella da rimuovere:

```
C:\TIZIO\CAIO>RD CARTELLIN
```

Ricordate che il comando DEL cancella i file, non le cartelle. Se una cartella ne contiene altre, anche se non contiene file, non risulta vuota, e non è eliminabile. Le cartelle contenute debbono essere (svuotate) e rimosse una ad una con il comando RD.

### **Visualizzare il contenuto di un file**

Se voglio vedere cosa c'è dentro un file, posso usare il comando TYPE (seguito da bianchi e dal nome del file). Se il file è lungo, scorrerà come un fulmine sul video fino all'ultima "pagina" (o quello che sia). Per fermare lo scorrimento durante la visualizzazione, potete provare a usare il comando di "ferma" (tasto "Scroll Lock", o qualche altra scritta tipo "Interr vis" "Bloc

Vis” o non so che altro. Cercatelo nella zona destra della tastiera, in genere è grigio). Con i computer contemporanei, dovete essere virtuosi della tastiera per potervene servire (provate, e capirete il perché).

Una alternativa è quello di chiedere al comando TYPE di fermarsi pagina per pagina e di aspettare un tasto. Si fa così:

```
C:\TIZIO\CAIO>TYPE SEMPRONIO|MORE←
```

( si aggiunge |MORE alla fine). Cercate il carattere “|” sulla tastiera. In genere è nello stesso tasto contenente la “\”, ma non è detto. E’ una barretta perfettamente verticale. Non lo confondete con il “!”. In gergo, |MORE si legge: “pipe-more”. Una spiegazione ci porterebbe troppo lontano: fino allo UNIX. Quindi, basta così.

Non tutti i file sono interessanti da guardare. Quelli, ad esempio, che hanno il suffisso EXE o COM (cioè programmi), oppure JPG o GIF (cioè fotografie o immagini), visti con il TYPE, (il quale stupidamente spara sul video tutto quel che dentro al file trova senza né decodificare né interpretare) sono piuttosto noiosi, e sembrano tutti uguali. All’incirca, così:

```
MZg~3÷+@ es+BhkÑ»Ñ , `kÑ»Ñ+óµB--ÑÑMSFT ¶|Ç|&|Gü -  
|+ $$$$$$$$$$$$$  
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$  
$$$$$$$$$$$$$$$$$$$$  
$$$$$$$$$$$$$$$$ &w . --`-.| T.ë-P.î R-  
PSQRVW 3Ä|++&ïE & t^+ J|RM|IC-/ü  
•CIuMü·MRuGë>d+î f+&ïE%="= &ïEu$| É+ | ç+ %=+u  
%a=au íl-;E w^AZ  
Y[X+•__SQRVWÄ <-&| D2&î F&î5÷ @-  
 3_Ä|+h&ïE & u __+ _|RM|IC -
```

Invece, farsi mostrare un file jpg da un apposito programma (che interpreta i dati contenuti nel file attraverso un algoritmo complesso in grado di restituirci l’immagine), può essere

tutt'altra cosa. Si omettono esempi, per i quali si fa affidamento sulla fantasia del lettore.

### **Copiare un file**

Ci si può domandare, di primo acchito, a cosa possa mai servire fare la copia di un file. Le risposte possono essere molte:

- Voglio portarmi via un file, per poterlo poi copiare sul computer in ufficio. Il file si trova sul disco rigido. Quindi, devo copiarlo nel floppy disk per potermelo portare via.
- Il file è così importante che ne voglio fare una copia (o più copie) in altre cartelle per essere assolutamente certo che qualunque cosa accada, almeno una copia rimanga intatta. Si tratta senza dubbio di un saggio proponimento, ma bisognerebbe riflettere sulla circostanza che una delle cose che possono accadere, e non tanto infrequentemente, oltre ad errate cancellazioni e confusioni tra file, è la rottura o cancellazione dell'intero disco rigido. Per prevenire le nefaste conseguenze di una tale evenienza, consultare il capitolo "Gestire il proprio calcolatore: norme sensate di comportamento".
- Sto facendo una delicata operazione di revisione, e voglio conservare copie delle varie versioni successive del file che sto modificando.

Altri motivi possono essere individuati, ma una cosa è certa, copiare file è indispensabile.

Per farlo, va usato il comando COPY, seguito dopo uno o più bianchi dal nome (o intero percorso, se necessario) del file da copiare, e dopo un altro bianco, dal nome (o percorso, se necessario) del file che deve contenere la copia.

Il nuovo file, contenente la copia, viene prima **creato**, e poi riempito con il contenuto del file da copiare. Quindi, non potete copiare un file su uno che già esiste: dovete prima cancellarlo (sempre che non lo abbiate chiamato nello stesso modo per errore). Ricorderete infatti che il DOS impedisce di creare un file

con lo stesso nome e percorso di uno già esistente (non vi possono essere file con lo stesso nome nello stesso posto!).

```
C:\TIZIO\CAIO>COPY SEMPRONIO CAIO
```

Crea una copia di “sempronio” in C:\TIZIO\CAIO con il nome di “caio”. Il nome del file è lo stesso della cartella che lo contiene, ma ciò è ammissibile, perché non vi è possibilità di confusione: **le due entità non stanno nello stesso posto**: la cartella CAIO sta nella cartella TIZIO, il file “caio” sta nella cartella CAIO.

```
C:\>COPY C:\TIZIO\CAIO\SEMPRONIO CAIO
```

Crea una copia di “sempronio” nella radice (COPY viene lanciato da lì!) e la chiama CAIO. Stando nella radice, è necessario riferirsi a sempronio con il percorso completo.

```
C:\>COPY C:\TIZIO\CAIO\SEMPRONIO  
C:\TIZIO\CAIO\CAIO
```

Ha il medesimo effetto del primo degli esempi fatti. Stando in radice, è necessario riferirsi ad entrambi i file con l'intero percorso (altrimenti si assumerebbe che entrambi debbano trovarsi in radice).

### **Cambiare nome a cartelle e file**

Se i nomi che avete scelto per file o cartelle non vi piacciono, potete cambiarli in ogni momento.

Basta usare il comando REN, seguito (dopo uno o più spazi) dal nome attuale della cartella, seguito ancora (dopo spazi) dal nuovo nome:

```
C:\TIZIO\CAIO>REN SEMPRONIO CAIO
```

### **Lancio dei programmi, la variabile PATH**

Lanciare in esecuzione un programmi (in DOS, i file che contengono programmi si distinguono per le estensioni EXE. COM

e BAT) è abbastanza semplice (almeno nelle circostanze usuali): basta, al prompt, digitarne il nome seguito da invio. Basta il prefisso, il suffisso può essere omesso.

Dato però che sappiamo che in DOS è possibile che si trovino, sullo stesso disco o su dischi diversi, due o più file con lo stesso nome (purché con percorso distinto), la sola "invocazione" (come si dice in gergo) del nome del file che contiene il programma potrebbe dare luogo ad ambiguità: se ci sono due file di programma con lo stesso nome, in posti diversi, qual è dei due che l'utente desidera effettivamente fare eseguire?

Se il file si trova nella cartella attiva, poco male: si potrebbe assumere che è quello il file desiderato. Ma se si trova altrove?

Analogamente a quello che si fa con il comando COPY per evitare ambiguità, se il file non è nella cartella attiva, si può digitare l'intero percorso. Il DOS eseguirà senza titubanze **quel** programma.

Ma costringere l'utente a digitare lunghe stringhe per invocare un programma o a dover navigare, a furia di comandi "CD", nel disco per trovarlo, (e a doversi ricordare a memoria dove si trova il programma in questione) non è esattamente quello che si dice un comportamento amichevole.

Per semplificare la vita è stato introdotto il concetto di "cammino prestabilito" per i programmi.

Per sapere quale sia questo cammino basta digitare il comando PATH ("Invio"). Si otterranno, se tale cammino è stato effettivamente impostato nel sistema, un elenco di percorsi separati da punto e virgola.

Se al prompt digitate il nome di un file (cioè, qualcosa di non identificabile in un comando) il DOS assume si tratti del prefisso (o di tutto il nome del file, se dopo un punto avete aggiunto una delle estensioni ammesse per i programmi) di un file contenente un programma. Se un file siffatto ("prefisso".BAT, oppure "prefisso".COM, oppure ancora "prefisso".EXE) si trova nella cartella attiva, il DOS lancia in esecuzione quel file. Se nella cartella attiva non è presente, inizia a cercarlo nella lista dei percorsi (in

ordine, da sinistra a destra). Appena lo trova, lo mette in esecuzione. Se, esaurita la lista del PATH, non lo ha ancora trovato, vi scrive sul video

File non trovato.

E vi restituisce il prompt, senza fare null'altro.

***Attenzione! Non lanciate programmi dei quali non conoscete il funzionamento. Potreste provocare il blocco della macchina e/o la perdita di alcune funzionalità e/o la perdita di dati!***

Come si fa a impostare il cammino prestabilito?

Chi lo definisce è un particolare file di programma, di tipo BAT, che viene eseguito automaticamente, come prima azione, appena il DOS parte, e che si chiama significativamente AUTOEXEC.BAT, e che in genere si trova in C:\. I file di programma BAT sono leggibili con il comando TYPE e contengono programmi (o algoritmi, se volete) espressi in un linguaggio piuttosto semplice: sono infatti righe di comando DOS che vengono eseguite una dopo l'altra. Utilizzando l'editor del DOS (Programma EDIT), è possibile esaminare e modificare AUTOEXEC.BAT (o qualunque file con estensione BAT). Naturalmente, per farlo, bisogna sapere quel che si sta facendo. Ma il percorso prestabilito è comunque definito in una riga dal seguente aspetto:

PATH=[Elenco di percorsi, separati da punto e virgola].

Volendo aggiungere un percorso, basta inserire un punto e virgola alla fine della riga e digitare di seguito il nuovo percorso (completo, a partire dalla radice inclusa) che conduce alla cartella che contiene il programma che si desidera venga trovato direttamente.

***Attenzione! Non tentate nemmeno di modificare il file AUTOEXEC.BAT se non siete sicuri di quello che state facendo. Potreste causare il blocco della macchina o la perdita di alcune funzionalità, o la perdita di dati!***

Spesso non è sufficiente indicare quale programma si desidera fare eseguire: magari questo programma agisce su di un file (modificandolo, ad esempio), ed occorre quindi indicare, nel momento stesso in cui si lancia il programma, su quale (o quali) file esso debba agire.

In questo caso, dopo il nome del programma, separati da esso e tra loro da spazi, si digitano i nomi (o gli interi percorsi, se necessario) dei file che il programma deve gestire. Si dice, in questo caso, che i nomi dei file vengono forniti al programma come “argomenti nella linea di programma”. Attenzione! Il DOS segue la scuola di pensiero di considerare, per distinguere il ruolo tra due o più argomenti, di prendere in considerazione l'ordine di dichiarazione. In altre parole, gli argomenti della linea di comando sono “posizionali”. State quindi attenti all'ordine in cui li digitate.

Esempio. Si supponga di possedere un programma, chiamato MAIUSCOL.EXE, che svolge la seguente operazione: prende in ingresso un file contenente un testo codificato in ASCII (in lettere maiuscole e minuscole) e lo trasforma nello stesso testo, ma tutto in lettere maiuscole. Una possibile “invocazione” di un tale ipotetico programma potrebbe essere:

```
C:\TIZIO\CAIO>MAIUSCOL SEMPRONIO SEMPRONIO.MAI
```

Il nostro ipotetico programma potrebbe volere come primo argomento il nome del file da trasformare, e come secondo il nome del file in cui depositare il risultato della sua operazione (in questo caso, lo stesso testo ma tutto in lettere maiuscole). At-

tenzione! Per quanto questo sia un modo tipico di operare dei programmi, l'esatto significato degli argomenti della linea di comandi dipende dal programma stesso, quindi, in ultima analisi, dalle scelte di chi quel programma ha concepito e sviluppato. Ci sono quindi "usanze", ma non regole certe.

*Per utilizzare un programma, bisogna conoscerne il funzionamento. Nei casi non ovvii, riferitevi al suo manuale d'uso!*

### **I file di testo. Modifica con EDIT**

Un modo particolarmente semplice di conservare un testo in un file, e di memorizzarvi la sequenza dei codici ASCII che lo compongono ("a capo" e caratteri bianchi inclusi). Un file siffatto prende genericamente il nome di "file di testo". In DOS, se non ha altri particolari significati, questo tipo di file ha normalmente il suffisso TXT (contrazione per TEXT).

Il comando di andare "a capo" è in realtà un carattere ("ritorno carrello", ASCII 13), che viene inserito premendo il tasto di "Invio" ("Enter", nelle tastiere USA), contraddistinto spesso dal simbolo ↵. Lo abbiamo già incontrato, con l'esatto stesso significato, come indicazione di "fine riga" dei comandi DOS.

Sono però file di testo, come detto poco prima, anche i file con suffisso BAT, che costituiscono in realtà un elenco di comandi DOS. I file BAT sono utili quando si desidera fare eseguire comandi DOS in successione e in modo ripetitivo, senza bisogno di ridigitarli ogni volta.

I file di testo possono essere "editati" (cioè, aperti, letti con comodità scorrendo le pagine sia in giù che in su, modificati e salvati nuovamente con il medesimo o con un nuovo nome) mediante appositi programmi che si chiamano genericamente "text editor". Il DOS fornisce un suo decente text editor, che si chiama EDIT. Una volta lanciato, EDIT funziona a menu, ed uno di

questi (la voce di menu "File", per l'esattezza) permette di cercare nel disco ed indicare quale file si desidera editare. Un altro modo di agire è quello di indicare il file come argomento della linea di comando, digitando, ad esempio:

```
C:\>EDIT AUTOEXEC.BAT
```

Oppure l'intero percorso del file da modificare, se necessario.

*Attenzione! Non tentate nemmeno di editare il file AUTOEXEC.BAT se non siete sicuri di quello che state facendo. Potreste causare il blocco della macchina, la perdita di alcune funzionalità o di dati!*

### **Altri comandi,**

Il DOS possiede altri comandi, ma essendo un sistema operativo in via di obsolescenza non è necessario un esame approfondito. Se servisse, sulla scorta delle spiegazioni fin qui date è facile, o utilizzando il comando HELP, o un libro apposito, o lo stesso manuale del DOS, approfondire il funzionamento dell'interprete di comandi del DOS.

Anche se obsoleto, ben difficilmente potrete essere esentati, almeno qualche volta, dal dovere utilizzare il DOS e qualcuno dei suoi comandi, almeno finché tale sistema sarà ancora usato. Avrete modo di accorgervene.

### **I sistemi operativi "a finestre"**

#### *Il vecchio mondo "a caratteri".*

Il breve excursus sul DOS appena compiuto vi avrà sicuramente fatto percepire che questo modo di operare non è tra i più comodi. Il contenuto delle cartelle non è sempre sotto l'occhio.

“Navigare” per il disco, magari per cercare un file che non si trova implica una notevole mole di digitazione (CD, DIR, CD, DIR... percorsi lunghi...).

Inoltre, come avrete notato appena lanciato un programma anche semplice come EDIT, il DOS esegue un solo programma alla volta. Gli dedica tutto il video, tutta la tastiera e, in generale, tutte le risorse. Se mentre scrivete una lettera volete consultarne un'altra, o altri dati gestiti da un altro programma, non potete farlo. Dovete *appuntarvi* (carta e penna!) quello che vi interessa prima di aprire il programma che volete utilizzare.

Il DOS è quello che si chiama un Sistema Operativo single task, cioè, un compito alla volta. Se dovete stampare qualche centinaio di pagine della vostra tesi, o del vostro ultimo poema in versi, prima di poter riutilizzare il calcolatore dovete aspettare che abbia finito.

Quel modo di stampare i dati sul video, inoltre, una riga dopo l'altra, come su una telescrivente, sembra poco consono al tubo a raggi catodici, che permette anche quella che si chiama “gestione a tutto schermo”, cioè la possibilità di muovere il cursore non solo dall'alto verso il basso, ma anche dal basso verso l'alto, e da sinistra verso destra, per andare a trovare l'informazione che si desidera sullo schermo e modificarla ovunque si trovi.

Non basta. L'operare per nomi (oltretutto contratti a sole otto lettere!) non è certo un buon modo per facilitare la memoria dell'utente, che è costretto a ricordare un mucchio di nomi (di programmi, di file dati, di cartelle) e a digitarli, spesso errando a causa della necessità di contrarre tutto in otto caratteri.

Si è cercato dunque di fornire agli utenti una interfaccia verso la macchina più amichevole, più ergonomica, più adatta al modo di pensare e di procedere degli esseri umani. In una parola, si è cercato di adeguare la macchina all'uomo, anziché il contrario.

## *Il nuovo mondo a icone e finestre.*

Le moderne interfacce a finestre (come X-Windows di Unix, MacOS di Apple, e, più recentemente, Windows e Windows95, 98 e NT, di Microsoft, tentano di rispondere a queste esigenze seguendo le linee identificate, sotto il profilo puramente teorico (non esistevano allora macchine in grado di fare tutto questo) molti anni fa alla Rank Xerox.

### **La scrivania (Desktop)**

L'idea base fondamentalmente è questa. Lo schermo del computer mostra una superficie colorata (metafora del piano della scrivania) sulla quale giacciono delle icone (pronuncia: icòna): piccoli simboli grafici che simboleggiano file contenuti nel sistema. Ogni file, oltre al suo nome, ha la sua icona. Le icone giacciono sulla scrivania, esattamente come gli oggetti che ci sono usuali: le cartelline con i nostri dossier, il blocco degli appunti, l'agenda, l'orologio, il calendario, ecc.

L'utente può toccare e usare questi oggetti in modo molto simile a come fa nella realtà: mettendoci "una mano" sopra. Quindi, deve avere un modo per indicare questi oggetti, piuttosto che digitarne il nome. Ecco nascere il mouse e il suo "puntatore" (metafora della mano, o del dito) che potete fare muovere sullo schermo-scrivania.

### **Le icone.**

Cosa potete fare con le icone (simulacri di "oggetti informatici"). Potete afferrarle e spostarle (muovere il mouse per portare il puntatore sopra l'oggetto, fare click e tenere premuto per "afferrare", muovere il puntatore che si trascina dietro l'icona, e la posate rilasciando il pulsante dove volete voi). Se volete disfarvi di quell'oggetto, fate come fareste nella realtà: lo trascinate su di un altro oggetto, il cestino, e ve lo depositate dentro.

## **Le finestre.**

Ma poiché una icona è il segnale di un oggetto informatico, quello che sicuramente vorrete fare è di “aprirla”, perché vi sveli il suo contenuto, esattamente come fareste con un dossier, o un libro, sulla vostra scrivania. Facendo doppio click con il mouse sopra l'icona (metafora del gesto di apertura) l'icona si apre trasformandosi in una finestra aperta sul “mondo” che l'icona rappresenta. Se si trattava di un archivio di numeri, vedete questi numeri. Se era una fotografia, attraverso (o “nella”) finestra vedete la fotografia. Altrimenti una lettera, un appunto, un elenco del telefono, ecc. Se l'icona era un programma, esso si mette in moto e la finestra contiene i mezzi necessari ad interagire con il programma.

Cos'è una finestra (window)? Se l'icona era un piccolo simbolo (disegnato in modo da evocarne facilmente il contenuto), una finestra altro non è che una apertura rettangolare (come le finestre) che occupa tutta o parte della scrivania. Come nel caso delle finestre, quello che ci si vede dentro dipende dal mondo che c'è dietro (dall'oggetto informatico che l'ha generata). Ma come in tutte le finestre vi sono bordi, cornici e maniglie per aprire e chiudere, anche nelle finestre informatiche vi sono alcuni elementi fissi. La cornice, uguale per tutte le finestre. Ma dato che non si tratta di finestre materiali, non c'è motivo che abbiano dimensioni fisse. Potete afferrare il bordo (click tenendo premuto con il mouse) e tirarlo, per modificarne le dimensioni. Da qualche parte una “maniglia” (pulsante) vi permette di “chiudere” definitivamente la finestra (farla tornare l'icona di partenza e basta). Un altro vi permette di ingrandirla d'un sol colpo fino a farla diventare grande come tutto lo schermo, se l'esame della situazione richiede di monopolizzare la vostra attenzione. Un altro pulsante vi permette di ridurla ai minimi termini (alle dimensioni dell'icona), senza però chiuderla, perché avete bisogno per un po' di spazio sul video per un'altra operazione.

Se il mondo sul quale la finestra è aperta è troppo grande per essere tutto contenuto, dovete avere qualche altra maniglia per

muovere la finestra nel mondo (oppure, se volete, per fare scorrere il mondo dietro la finestra) in modo da poter esaminare, di quel mondo, l'angolo che in quel momento vi interessa. Ecco quindi che, in questi casi, compaiono lungo il bordo della finestra delle barre di scorrimento, con pulsanti da afferrare o da premere per far muovere il mondo, dietro la finestra, in alto, in basso, a destra o a sinistra. Potete così scorrere un libro, avanti e indietro per le sue pagine, o far camminare la vostra finestra in giro per una mappa.

Non basta. L'interazione con gli oggetti informatici non è generalmente passiva. Aperto un documento, normalmente volete modificarlo, per aggiungervi qualcosa o per correggerlo. Così dicasi per un disegno o qualsiasi altra cosa. Per fare questo, e talvolta anche per potere agevolmente consultare dei dati, sono necessarie due cose: scegliere su cosa si vuole agire, e impartire un comando che indichi l'azione che su quanto scelto si vuole intraprendere. Nelle interfacce a carattere, questo si risolve con comandi digitati, ma noi non vogliamo più costringere l'utente a defatiganti *tour de force* sulla tastiera. Vogliamo che possa *scegliere* ciò che desidera da elenchi precostituiti, in modo che non debba *ricordarsi* come si faccia a fare la tal cosa. Dunque, deve scegliere da un *menu* di operazioni possibili.

Quindi, dobbiamo dare all'utente un modo di *selezionare* la porzione di mondo su cui agire, semplicemente indicandola con il mouse (facendo click e trascinando, per estendere la selezione dal punto iniziale fino a dove si vuole operare la scelta), e infine il modo di scegliere quale azione vada intrapresa sulla selezione.

### **La selezione e i menu.**

Ecco allora, nella parte superiore delle finestre, una striscia dedicata ad ospitare i menu. I menu si presentano generalmente come elenchi di voci, ciascuna con un titolo che ricorda l'argomento delle funzioni. Facendo click su di un argomento, da questo discende un elenco di sottoargomenti più dettagliati,

facendo click sul quale si scende ancora di dettaglio fino ad arrivare al comando vero e proprio, che a questo punto, facendovi click sopra, viene eseguito su quanto selezionato.

Se tutto questo sembra (per ora) un po' astratto, pensate, ad esempio, alla necessità di cambiare il colore di una frase in un testo. Bisogna muovere la finestra sul testo, finché non si è trovata la frase. Bisogna indicare esattamente la porzione di testo (la frase) sulla quale si intende operare, facendovi scorrere sopra il puntatore del mouse, e infine bisogna scegliere, nei menu, il comando di definizione del colore del testo.

Dividendo i comandi in menu strutturati (argomenti e sottoargomenti) è possibile scegliere tra letteralmente migliaia di comandi possibili non solo non dovendoli ricordare uno per uno, ma potendoli ritrovare anche rapidamente, senza necessità di scorrerli tutti.

### **Più compiti, più finestre contemporaneamente. Il multitasking.**

Anche questo però non è ancora sufficiente. Raramente un lavoro consiste nella consultazione di un solo documento, o fonte di dati, alla volta. E' quasi sempre necessario tenere sott'occhio un paio di documenti distinti, e magari l'agenda, caso mai qualcuno ci telefonasse nel frattempo.

Ecco allora che il sistema permette di aprire più finestre contemporaneamente. Non solo. Se una finestra rappresenta un programma che deve fare qualche lavoro, esso lo deve eseguire senza né impedire all'utente di continuare ad aprire, chiudere, spostare le icone e le finestre che desidera, ma senza impedire che altre finestre di altri programmi come lui cessino di funzionare in attesa che lui abbia finito (ricordate il DOS? Un solo programma alla volta).

Quindi il sistema deve essere, come si dice, multitasking, permettere lo svolgimento contemporaneo di più compiti indipendenti e la consultazione di più finestre.

### **Navigazione tra le finestre aperte e scelta di quella attiva.**

Naturalmente, tutte queste finestre aperte possono affollare lo schermo. Possono finire una sopra all'altra, in tutto o in parte. Viene naturale pensare che risultino, in questo caso, impilate una sopra all'altra, e che risulti totalmente visibile solo quella in cima alla pila. In gergo, si dice "la finestra attiva" oppure "la finestra con il focus". Le altre spunteranno, se spuntano, sotto a quest'altra. Viceversa, se metto due o più finestre una accanto all'altra, in modo che non si coprano, devo potere vedere il loro contenuto contemporaneamente.

Deve essere però mantenuto il concetto di "primo in alto nella pila". Infatti, se sposto le finestre fino a sovrapporsi, una deve finire "sotto" e l'altra "sopra". In ogni istante, una sola finestra è in cima alla pila. Devo però avere un mezzo semplice per spostare un'altra finestra in cima alla pila. Se ne spunta da qualche parte un pezzo, e questo la rende riconoscibile, posso farci click dentro per indicare che la voglio spostare sopra. Essa finirà dunque sopra, occultando le altre e mostrando per intero il suo contenuto.

Ma se la finestra è completamente nascosta da un'altra, oppure il pezzo che spunta non è sufficiente a riconoscerla, devo avere un altro mezzo per spostarla in cima alla pila. Qualcosa come un "menu di scrivania" che elenchi le finestre aperte, o qualcosa del genere.

Ecco allora la famosa barra inferiore di windows, nella quale compare un pulsante per ogni finestra aperta.

### **Copia, taglia e incolla.**

Quando si lavora con i documenti cartacei, capita spesso di fare dei collage. Ho bisogno di una figura presa da un documento per inserirla in un altro, oppure nello stesso ma in un altro posto. In questi casi, se il documento originale non mi interessa più, posso operare con le forbici e la colla: taglio la figura e la incollo nel suo nuovo posto. Se invece devo incollare una co-

pia, faccia una fotocopia e ritaglio quella per incollarla. Chiamiamo queste operazioni taglia e incolla” (cut and paste) oppure “copia e incolla” (Copy and paste).

Tornando al nostro mondo “virtuale” di finestre, questo vuol dire che devo poter fare taglia o copia e incolla non solo fra parti diverse dello stesso documento, ma anche tra documenti diversi o addirittura di natura diversa.

In altre parole, devo poter fare “taglia o copia e incolla” anche tra finestre diverse, possibilmente senza preoccuparmi nella natura “tecnica” dell’oggetto informatico che c’è dietro, ma guardando solo ciò che nella finestra si vede: se ciò che vedo mi piace, devo potere avere la possibilità di *definire un’area di taglio o di copia*, con un attrezzo simile concettualmente ad un paio di forbici (non distruttive, nel caso del “copia”), poter dare un comando di copia o taglia, spostare il focus sulla nuova finestra, indicare *dove* voglio venga incollato, e impartire finalmente il comando di “incolla”. Subito devo poter vedere quanto appena incollato nel suo nuovo posto, in mezzo a quanto già c’era nella finestra.

Tutti i Sistemi Operativi a finestre attuali permettono il taglia, copia e incolla fra finestre diverse.

### **Programmi e dati.**

Infine, un’ultima cosa dalla quale desideriamo l’utente sia sollevato. Sappiamo che un file può contenere dei dati, o dei programmi (per gestire questi dati). Sappiamo che in merito, chi fa programmi, può fare ciò che vuole. Può stabilire che i dati che il suo programma gestisce vengano memorizzati secondo formati qualsivoglia, che solo il programma in questione è in grado di interpretare e gestire correttamente. Ricordate il TYPE sotto DOS di una immagine JPG? Una sequenza di dati incomprensibili, tali comunque da non comporre una immagine sul video. Se utilizzate invece del TYPE (che interpreta tutto come codifiche ASCII, aspettandosi un file semplice di testo) un program-

ma appositamente, questo vi stamperà sul video la corrispondente fotografia.

Ogni file dati ha bisogno del suo specifico programma per essere correttamente gestito.

In DOS, è l'utente che deve saperlo, che deve ricordarsi che quel file è di quel tipo e deve essere aperto con il tale programma.

Ora si vorrebbe che l'utente non debba necessariamente ricordarsi tutto ciò (oramai, i programmi sono decine di migliaia!) ma sia il sistema stesso a memorizzare, per ogni file dati, quale sia il programma opportuno da utilizzare per la sua gestione.

Così facendo, l'utente deve occuparsi solo delle cose che lo concernono: cioè, i dati, e i relativi file. Dato che si tratta di informazioni sue, queste le deve necessariamente gestire lui. Ma quanto al fatto di sapere quale sia il programma adatto a gestire quei dati, non è più necessario lo ricordi. Indicando (doppio click, metafora dell'apertura della finestra) quale sia il file (icona) desiderato, il sistema si incaricherà di lanciare l'opportuno programma facendogli aprire il file scelto.

### *I sistemi operativi a finestre reali.*

Tutte le caratteristiche e i desiderata che abbiamo elencato sono stati soddisfatti in maniera più o meno totale o brillante dai sistemi operativi a finestre oggi esistenti. Ognuno brilla per qualcosa (qualcuno non brilla affatto) e meno per qualcos'altro.

Vediamo in dettaglio: oggi troviamo disponibili X-Windows (basata sul sistema operativo UNIX, a torto o ragione poco utilizzata nella elaborazione personale). MacOS, per il McIntosh Apple. NextStep, per workstation, oggi defunto ma probabilmente resuscitato da Apple come prossima generazione di Sistema Operativo per McIntosh. Infine, Microsoft., con la sua Windows 3.1X, Windows 95, 98, ME, Windows NT e Windows 2000.

### **Luci ed ombre: McOS**

L'implementazione della metafora di scrivania, icone e finestre è particolarmente completa nel McOS, nonostante sia un sistema operativo piuttosto "anziano". Fin dall'inizio le restrizioni sintattiche ai nomi degli oggetti sono state ben poche: nomi lunghi (fino a 64 caratteri) con la possibilità di usare lo spazio bianco. Quindi, i nomi delle cose possono essere quelli veri, della vita di tutti i giorni. La foto della nipote si può chiamare "mia nipote Elena al mare nell'agosto del 1994". L'associazione automatica tra file dati e applicazioni è risolta molto bene, senza limitazioni di alcun genere, ed è individuale, file per file.

L'implementazione del multitasking è invece un po' da giocattolo. Si tratta di quello che si chiama "multitasking cooperativo", cioè un multitasking per modo di dire, nel quale la macchina può restare bloccata dall'esecuzione di un compito particolarmente gravoso (come una stampa, ad esempio). Altre ombre riguardano l'assenza di isolamento dei programmi, per cui un errore in un programma può mandare in blocco o in crash ("bombe", nel linguaggio Mac) tutto il sistema.

### **Luci ed ombre: Windows 95-98-ME**

Non parliamo nemmeno dell'antesignano Windows 3.11 (orami giustamente dimenticato), che non implementava nemmeno una vera metafora di scrivania. Nella generazione successiva di Windows per uso domestico, la metafora di scrivania, icone e finestre è ben implementata. Il legame tra oggetti sulla scrivania e file sul disco è abbastanza naturale: ogni oggetto è un file, e ogni file può diventare un oggetto semplicemente trascinandolo sulla scrivania. E' comparso anche il cestino (dove potete frugare per recuperare qualcosa che avevate gettato via, se vi pentite, purché non aspettiate troppi giorni. Cosa presente in McOS fin dalle origini, ma meglio tardi che mai. Il multitasking è di tipo preemptive (vero e proprio, quindi). Nessun compito (o quasi nessuno) è in grado di bloccare completamente la macchina

impedendo le altre funzionalità. Purtroppo il meccanismo di associazione automatica tra file dati e programmi (inspiegabilmente) è rimasto quello di Windows 3.1X, cioè è per famiglie, ciascuna identificata dal suffisso del file, e quindi soffre degli stessi inconvenienti dell'antenato. In compenso le limitazioni sui nomi dei file sono cadute (255 caratteri spazio bianco incluso) e quindi si può tornare a chiamare la foto della nipote con il suo vero nome. L'isolamento delle applicazioni è insufficiente, e quindi non di rado errori dei programmi applicativi possono determinare il crash del sistema.

#### **Luci ed ombre: WindowsNT (versione 4.0)**

Si tratta di un sistema operativo degno effettivamente di questo nome, che ha decisamente perso quella soffusa aura da giocattolo caratteristica, sia pure in misura diversa, di quelli fin qui citati. L'interfaccia utente è identica a quella di Windows95, e con essa condivide purtroppo il rudimentale meccanismo di associazione tra file dati e programmi, attraverso i suffissi, con tutti i suoi limiti. Peraltro, oltre ad essere un vero multitasking, garantisce il completo isolamento delle applicazioni, per cui è virtualmente impossibile, per un programma applicativo, provocare crash di tutto il sistema. Per quanto mal concepito e mal fatto, un programma applicativo non può bloccare nessuna delle altre funzionalità del sistema, né il funzionamento degli altri programmi. Con il suo file system nativo gestisce anche un complesso sistema di sicurezza e protezione dei dati, essendo di fatto un sistema multi utente.

#### **Luci ed ombre: NextStep**

Si tratta di un sistema operativo serio e affidabile, multitasking e multiutente, con una corretta gestione del disco e delle applicazioni, basato sulla lunga tradizione UNIX. Nato su macchine particolari (Next), è stato poi portato su Workstation SUN. E' morto per ragioni essenzialmente commerciali che nulla hanno

a che vedere con la qualità del prodotto. Continua a conservare un interesse perché è diventato la base per il prossimo sistema operativo della Apple per i McIntosh. L'operazione di trasformazione di un programma da una macchina all'altra prende il nome di "porting". In questo caso, non è ancora noto cosa del sistema originale sopravviverà nel porting verso il McIntosh.

### *I sistemi operativi gratuiti – il gruppo GNU*

Può sembrare a prima vista incredibile, ma nell'informatica esistono solide, attive e prolifiche associazioni senza scopo di lucro che producono il cosiddetto software "freeware" (gratuito), al solo scopo di permettere la diffusione dell'informatica anche tra i "non abbienti" e contrastare le (forti) tendenze monopolistiche del settore.

Cuore di questo genere di iniziative è GNU, un gruppo che raccoglie molti contributi universitari anche molto qualificati (Stanford University, ad es.). Tra le tante cose che GNU ha messo a disposizione gratuitamente del mondo, anche alcuni sistemi operativi.

Tra questi, LINUX, una implementazione (e oggi anche molto completa e molto affidabile) di UNIX su computer IBM compatibili. Il problema di LINUX è che non ci sono molti applicativi di quelli "diffusi", come elaboratori di testi e fogli elettronici. In compenso è molto usato come sistema server per Internet, utilizzando un programma Web Server anch'esso gratuito chiamato "Apache" (Senza LINUX e Apache, Internet si sarebbe così diffusa nel mondo?).

L'iniziativa forse più interessante è ancora in corso di sviluppo, ed è partita da meno di un anno. GNU ha raccolto un gruppo di circa 2000 specialisti (che collaborano via Internet) per sviluppare "Freedows", ovvero Windows gratuita. Freedows dovrebbe garantire la completa compatibilità non solo con Windows, Windows95, WindowsNT, ma con virtualmente tutti i sistemi operativi a finestre oggi esistenti (incluso NexStep). L'iniziativa

ha il dichiarato scopo di contrastare il monopolio Microsoft nel campo dei Sistemi Operativi, mettendo a disposizione un sistema operativo compatibile, concorrente, e gratuito.

Se ne potrà parlare quando uscirà, ma GNU è un gruppo altamente professionale, motivato, e che produce software di alta qualità. Avremo modo di tornarci a proposito di Aladdin GhostScript e GhostView, visualizzatore gratuito di file PostScript.