La candela

Eravamo rimasti a metà di un discorso, che ora riprendo. Ricordo che il tema centrale è il riduzionismo, ma sto sviluppando un'illustrazione della struttura e del funzionamento dei computer, che — come ho già accennato — mi servirà per spiegare e motivare la mia tesi a proposito del riduzionismo di cui sopra.

La veloce panoramica ha percorso la struttura dei computer dal basso, e fino a questo punto ha portato a una tabella, o meglio una scala, parallela all'analoga di Popper che ho citato all'inizio della puntata precedente:

4. Integrati a grande scala: CPU, memorie RAM
3. Integrati a media scala: addizionatore, contatore ...
2. Circuiti integrati a piccola scala
1. Flip-flop, porte logiche
0. Transistors, diodi, resistenze....

A proposito della CPU avevo scritto: "la sua funzione è d'interpretare ed eseguire le *istruzioni*, seguire l'andamento di un *programma*, ricevere dalla memoria i dati e rinviare i risultati di ogni operazione che esegue." Dunque, ripartiamo da qua.

Un computer non sarebbe un computer se non eseguisse istruzioni, organizzate in un programma. Non posso scendere in maggiori dettagli, se non questo: il programma è un insieme d'istruzioni, abbiamo detto. Esso si trova conservato nella memoria, da dove la CPU lo legge, un'istruzione dopo l'altra. L'esecuzione delle istruzioni richiede il trasferimento di dati dalla memoria alla CPU e viceversa (e sto semplificando in modo quasi osceno: per es. trascuro l'esistenza delle "memorie di massa," delle "periferiche" ...).

Ma a me interessa che parlando d'istruzioni e programma abbiamo scalato un altro livello, e di nuovo ci troviamo nella situazione già vista:

- da un lato, istruzioni e programma sono facilmente riducibili ai livelli inferiori
- dall'altro è non solo utile ma necessario, nell'uso di un computer, ignorare tale riduzione, e considerare invece le istruzioni come entità primitive *astratte*; un programma come un insieme d'istruzioni che si può studiare *a prescindere* dall'effettiva realizzazione in un computer.

Qui è comparsa una nuova parola chiave: "astratto." In realtà il processo di astrazione è già avvenuto ai livelli precedenti, sebbene se non ci avessi richiamato l'attenzione: anche considerare un flip-flop in senso funzionale, trascurando il modo come opera in termini di componenti elementari, è un'astrazione. Si può realizzare un flip-flop in più modi equivalenti, e allo stesso modo si possono avere

computer che eseguono *lo stesso programma*, anche se sono fisicamente distinti. Anzi, nasce qui un nuovo concetto: quello di *emulazione*. Lo si usa per indicare che un certo computer può essere programmato in modo da emularne un altro, sì che il primo si comporti in modo indistinguibile dal secondo, pur essendo del tutto diverso ai livelli inferiori.

Aggiungiamo dunque il nuovo livello alla nostra scala:

5. Istruzioni, programma in senso astratto
4. Integrati a grande scala: CPU, memorie RAM
3. Integrati a media scala: addizionatore, contatore ...
2. Circuiti integrati a piccola scala
1. Flip-flop, porte logiche
0. Transistors, diodi, resistenze....

Ma in realtà potrei dire che a questo punto siamo entrati in un altro mondo: quello appunto della programmazione. Programmare un computer al livello a cui siamo è cosa lenta, difficile, e soprattutto non permette grandi passi avanti. Per questa ragione fin da subito si sono escogitati metodi per rendere il lavoro di programmazione più "a scala umana."

> "La giornata dopo fu molto sgradevole: pioggia freddo e nebbia sottile sulla città di Cambridge. Kingsley lavorò tutta la mattina e tutto il pomeriggio davanti a un fuoco scoppiettante, nel suo appartamento all'Università. Lavorava sodo, tracciando sulla carta degli strani scarabocchi di cui diamo un breve esempio, un esempio cioè del codice grazie al quale si poteva ordinare al calcolatore come compiere i calcoli e le operazioni necessarie:

	T		Z
0	A	23	Θ
1	U	11	Θ
2	A	2	F
3	U	13	Θ

> "Verso le tre e mezza uscì dall'Università ben imbacuccato e riparando sotto l'ombrello un voluminoso fascio di carte. [...]"

Dubito che qualcuno riconosca questa citazione. L'ho presa da un romanzo famoso: *La nuvola nera* di Fred Hoyle (1958). Il nome di Hoyle vi è certo noto, come astrofisico teorico, ma forse anche come scrittore di fantascienza. *La nuvola nera* è — se non erro — la sua prima prova. Il protagonista, Kingsley, è più o meno un *alter ego* dell'autore, e qui viene presentato alle prese con la programmazione di un tipico computer del tempo, l'EDSAC, che entrò in funzione a Cambridge nel 1949. Quando lessi il romanzo conoscevo bene l'EDSAC

e la sua programmazione, per il lavoro che stavo facendo alla CEP; per cui riconobbi subito il codice, anche se Hoyle non lo spiega, e se oggi l'ho dimenticato completamente...

Lo scopo di questo piccolo esempio era appunto di mostrare quanto fosse noioso e complicato allora programmare un computer. Considerate che un calcolo anche modesto di righe di quel genere ne poteva contenere delle centinaia, e guai a sbagliare...

Ecco perché la necessità di portare la programmazione più vicina alla scala umana: il primo passo furono i cosiddetti *linguaggi simbolici*, sui quali non dico niente per brevità; ma subito nacquero i linguaggi *di alto livello* (il primo fu il FORTRAN, 1957) il cui scopo era ed è quello di far lavorare il programmatore in un modo il più vicino possibile al linguaggio della matematica, ovviamente familiare al fisico come all'ingegnere.

Posso dunque aggiungere altri due gradini:

7. Linguaggi di alto livello (FORTRAN, C ...)
6. Linguaggi simbolici
5. Istruzioni, programma in senso astratto
4. Integrati a grande scala: CPU, memorie RAM
3. Integrati a media scala: addizionatore, contatore ...
2. Circuiti integrati a piccola scala
1. Flip-flop, porte logiche
0. Transistors, diodi, resistenze....

Ancora una volta una riflessione: il passaggio ai linguaggi di programmazione esprime un ulteriore distacco dell'utilizzatore dalla struttura del computer. Ora non solo non interessa come siano realizzate la memoria o la CPU, ma non occorre neppure sapere che esistono: per il programmatore esistono solo dati, variabili, espressioni che indicano in forma matematica operazioni da eseguire.

È un altro passo di astrazione, ma al tempo stesso è la condizione necessaria per poter affrontare (e quindi far risolvere al computer) problemi di un grado di complicazione altrimenti inaccessibile.

* * *

Ma deve ancora entrare in scena un altro attore, senza del quale il computer (esclusi al più i primi e primitivi) non potrebbe funzionare o almeno non potrebbe essere usato: il *sistema operativo* (SO). Un SO è sempre all'opera non appena accendiamo il computer, e potrei forse dire che la sua funzione consista nell'agire nel modo più invisibile. Per capirlo, debbo ricordare che finora ho lasciato da parte componenti essenziali di qualsiasi computer: le *memorie di massa* e le *periferiche*. Nei computer di oggi le memorie di massa s'identificano principalmente coi dischi fissi o coi CD; periferiche ce n'è dei più diversi tipi, ma alcune sono indispensabili: la tastiera, lo schermo video (spesso detto "monitor"). Aggiungiamo il mouse, la stampante, il modem, lo scanner, la scheda audio...

Bene: il funzionamento di tutti questi apparati è reso semplice grazie all'opera del SO. Dipende dal SO l'organizzazione dei nostri documenti in "cartelle" (files), che possiamo aprire, chiudere, copiare, con semplici comandi in ogni linguaggio di alto livello (e addirittura con un click del mouse su un bottone, ma di questo fra un po').

Ma anche un'operazione apparentemente così banale, come premere il tasto "a" sulla tastiera, mette in realtà in azione un complesso di programmi, i quali fanno sì che

a) il SO si accorga che è stato premuto un tasto, e precisamente quale tasto

b) venga trasmesso al monitor, attraverso un programma apposito (*driver*) il giusto insieme di segnali che fanno accendere e spegnere proprio quei pixel che nel loro insieme formano, nella giusta posizione, la lettera "a" che io vedo sullo schermo

c) l'informazione che è stato premuto quel particolare tasto venga trasmessa ad altri programmi, per es. al programma di scrittura con cui sto scrivendo l'attuale puntata della "Candela."

Chiunque capisce quanto sia importante che tutto questo lavoro venga svolto in modo invisibile, affinché l'utente non se ne debba preoccupare, anzi non ne sia neppure consapevole. Salvo quando qualcosa non funziona, e allora sono guai...

Allo stesso modo, è assai utile, anzi necessario, che nel normale lavoro l'utente non debba sapere né preoccuparsi di come realmente i suoi documenti vengono conservati su un disco fisso: anche qui c'è un complicatissimo gioco di segnali magnetici, che a livello più alto diventano caratteri "scritti" su disco, e che nel loro insieme formano parole, e poi questo testo che sto scrivendo. In qualche posto, nei visceri del computer, i miei pensieri diventano magnetizzazioni di un disco che ruota; ma io non ne so nulla, e penso di aver "scritto" delle pagine, come se fossero fogli di carta conservati in una cartella di un classificatore.

È proprio il fatto che io non ne sappia nulla (anche quando, com'è il mio caso, qualcosa ne so, ma *debbo* far come se non ne sapessi) che mi permette di concentrarmi su quello che voglio scrivere, bello o brutto che sia.

Va anche detto che questa "trasparenza" del SO non è sempre senza inconvenienti: ne sa qualcosa quel terrorista che credeva di aver cancellato dal suo computer dei files compromettenti, fidando nella metafora "file = fogli di carta," e non sapeva che di regola un SO non cancella *fisicamente*, per cui i dati possono essere ancora accessibili a un vero esperto. Magari il terrorista si reputava un esperto: peggio per lui. Forse aveva anche la "patente europea del computer," che in realtà non produce affatto degli esperti, ma solo (pagando fior di quattrini) degli utilizzatori terra-terra dei prodotti Microsoft. Ma questo tema ci porterebbe lontano, e basti dunque l'allusione...

Intanto la scaletta si arricchisce di altri due gradini:

8. Sistemi operativi
7. Linguaggi di alto livello (FORTRAN, C ...)
6. Linguaggi simbolici
5. Istruzioni, programma in senso astratto
4. Integrati a grande scala: CPU, memorie RAM
3. Integrati a media scala: addizionatore, contatore ...
2. Circuiti integrati a piccola scala
1. Flip-flop, porte logiche
0. Transistors, diodi, resistenze....

* * *

Più o meno insieme coi primi SO, è nata un'altra categoria di "oggetti" rivolti agli utilizzatori di computer: quelli che col tempo avrebbero assunto la denominazione di "programmi applicativi," o brevemente di "applicazioni." Di che si tratta? Mr. Jourdain, voi avete sempre parlato in prosa, senza saperlo... Voglio dire che tutti voi che fate uso di computer, non fate altro che usare programmi applicativi, dei tipi più diversi: programmi di scrittura, di posta elettronica, di elaborazione d'immagini; fogli elettronici, programmi per la presentazione pubblica di documenti, per l'ascolto e registrazione di musica ... e molti altri.

Alcuni dei più antichi hanno oltre vent'anni, mentre altri non avrebbero potuto nascere senza le prestazioni (soprattutto grafiche) delle macchine più recenti; ma tutti si appoggiano pesantemente su un sistema operativo, che si assume il "lavoro duro," a noi invisibile. Noi scriviamo lettere, ritocchiamo foto, elaboriamo un bilancio ... e come al solito non sappiamo niente di quello che succede dietro le quinte. E per fortuna è così: se provassimo a pensarci, ci perderemmo la testa, mentre invece ci limitiamo a correggere o spostare una frase, a cambiare la tonalità dominante della foto, a modificare la formula per operare sulle righe o le colonne del foglio elettronico.

È una nuova astrazione, che a stretto rigore non dovrei mettere sopra a quella del SO, ma neppure sotto. Ma dato che una scala è per sua natura unidimensionale, opto per il sopra, ed ecco la nuova versione:

9. Programmi applicativi
8. Sistemi operativi
7. Linguaggi di alto livello (FORTRAN, C ...)
6. Linguaggi simbolici
5. Istruzioni, programma in senso astratto
4. Integrati a grande scala: CPU, memorie RAM
3. Integrati a media scala: addizionatore, contatore ...
2. Circuiti integrati a piccola scala
1. Flip-flop, porte logiche
0. Transistors, diodi, resistenze....

* * *

Nella storia ormai semisecolare dei computer, di SO ne sono stati inventati parecchi (anche se oggi quasi tutti conoscono soltanto Windows e credono che un computer non si possa usare altro che così). A partire dai primi anni '80 c'è stata un'evoluzione importante: è nata la *metafora della scrivania* (desktop) e delle finestre, che simulano i diversi fogli di carta che si trovano su qualunque scrivania. In parallelo, ha fatto la sua comparsa quel simpatico animaletto, il "topo," che naturalmente noi chiamiamo "mouse" come se fossimo parlanti inglese, mentre per es. i nostri "cugini" francesi lo chiamano da sempre "souris" (ovvero sorcio, topo appunto).

Dato che sono sicuro che su finestre, cartelle, e altri simili aggeggi "informatici" ne sapete almeno quanto me, non mi ci soffermo, se non per rilevare che siamo davanti a un nuovo livello di astrazione. Mentre nei "vecchi" SO per es. per copiare un file da un posto a un altro dovevo scrivere qualcosa come

```
copy candela42.tex finito
```

ora mi limito a "trascinare" l'icona del file da una finestra a un'altra sullo schermo. Mentre prima per stampare il mio file dovevo battere sulla tastiera poniamo

```
print candela42.ps
```

ora mi basta "cliccare" sul bottone che porta l'icona della stampante, ecc. Grande vantaggio soprattutto per chi non è a suo agio con una tastiera e scrive solo con due dita; ma il punto importante è che così facendo ci allontaniamo ancora di un altro passo dal computer. Non abbiamo più neanche bisogno di sapere che per copiare il nostro SO richiede il comando "copy" e per stampare il comando "print." Ci sono meno cose da ricordare, si fa più presto... Ci sono anche degli inconvenienti, ma non mi ci voglio fermare, perché come sapete il mio scopo è un altro.

In sostanza, la nostra scaletta ha acquistato un altro gradino, e ha preso la forma finale (per ora: domani chissà ...)

10. Metafora della scrivania, mouse, ecc.
9. Programma applicativi
8. Sistemi operativi
7. Linguaggi di alto livello (FORTRAN, C ...)
6. Linguaggi simbolici
5. Istruzioni, programma in senso astratto
4. Integrati a grande scala: CPU, memorie RAM
3. Integrati a media scala: addizionatore, contatore ...
2. Circuiti integrati a piccola scala
1. Flip-flop, porte logiche
0. Transistors, diodi, resistenze....

Solo una piccola notazione sull'ultimo passo. Ho detto che è avvenuto circa 20 anni fa, ma la sua vera diffusione è molto più recente. Come mai, visto che ha chiari vantaggi? Il fatto è che per far lavorare un computer a quel modo occorre una potenza di calcolo (velocità, capacità di memoria) e una qualità della presentazione grafica che non erano disponibili, o lo erano a carissimo prezzo, quando l'idea nacque. Qui vediamo come il progresso tecnologico nello hardware (ai livelli più bassi) sia stata condizione necessaria per un passo di astrazione su un livello parecchio distante.

* * *

E ora, cominciando a riprendere il discorso generale sul riduzionismo, chiediamoci anzitutto: che ne è dell'idea di Popper a proposito della causalità verso l'alto o verso il basso? Ricordo che secondo Popper una riduzione ai livelli inferiori è possibile a condizione che non ci sia causalità verso il basso: che i livelli superiori non possano agire su quelli inferiori. Avevo già criticato quest'idea nella sua applicazione al mondo fisico; ma come stanno le cose nel mondo dei computer? Nel caso del computer, vi sono indicazioni di una causalità verso il basso? Se sì, e se Popper avesse ragione, allora ne dovremmo concludere che anche in un computer il programma riduzionista non è realizzabile...

Ma che in un computer ci sia causalità verso il basso è piuttosto ovvio: ne ho già dato implicitamente un esempio quando ho descritto che cosa accade se si preme un tasto. Non solo entrano in funzione programmi (livello 5), ma qualcosa viene scritto nella memoria (livello 4) e in ultima analisi qualche transistor si mette in conduzione, qualche condensatore si carica o si scarica (livello 0)... Non ho bisogno di spiegare in dettaglio che cosa capita se trascino un'icona sullo schermo: a questo punto potete forse immaginare (vedere?) programmi che girano, istruzioni che cambiano il contenuto di registri, tensioni che salgono e scendono, pixel che si spengono o si accendono in vari colori...

Dunque un computer non è "riducibile"? Ma certo che lo è! L'esempio l'ho scelto apposta! Posso dire di aver dimostrato un teorema? *L'interpretazione di Popper a proposito del riduzionismo è sbagliata.*

Però ... però il discorso non è mica finito... Ora debbo rendere esplicito ciò che fin qui era rimasto solo accennato, affidato a delle note qua e là. Le annotazioni che voglio richiamare sono sostanzialmente di tre tipi:

a) Ho fatto notare più volte che *non occorre*, oltre a essere *assai complicato*, conoscere in dettaglio il livello inferiore per poter lavorare su quello superiore.

b) Ho anche detto in varie occasioni che questa *astrazione* è anzi *necessaria*: che se si restasse attaccati ai dettagli, non si potrebbe ragionare a un livello più alto, non si potrebbero risolvere problemi, anzi neppure formulare nuovi concetti.

c) Ho sottolineato che addirittura se ci si sforzasse ogni volta di ridursi ai livelli più bassi, si finirebbe per perdersi, per non capire niente.

Ci sono dunque diverse ragioni per non perseguire la completa riduzione: ragioni pratiche (anche se possibile, sarebbe terribilmente complicato); ragioni cognitive (non riusciremmo a tenere parallelamente insieme i diversi livelli); infine ragioni che potrei chiamare teoretiche: non potremmo formulare concetti a un grado di astrazione superiore.

Tutto questo, non dimentichiamolo, l'ho ricavato da un esempio scelto su misura: quello del computer. Si tratta di un esempio felice, perché da un lato ci libera dal dubbio che l'eventuale non riducibilità possa dipendere dalla natura del sistema: qui abbiamo a che fare con qualcosa di perfettamente conosciuto, nei minimi dettagli, in quanto concepito e costruito sulla base di conoscenze già esistenti.

Dall'altro, lo sviluppo dei computer è un fatto recente, che ha preso solo una generazione: sono ancora vive persone che erano già sulla breccia, in senso scientifico, quando i computer letteralmente non esistevano. Perciò anche la storia di questo sviluppo ci è perfettamente nota, addirittura nella memoria di alcuni di noi. E in particolare questo sviluppo mostra l'interazione fra le prestazioni dello hardware, in termini di velocità e di complessità, e la nascita dei successivi gradini della scala: senza abbastanza memoria e CPU abbastanza veloci, niente SO complessi, niente finestre grafiche...

Ma qui s'inserisce la mia tesi: dal momento che vediamo accadere questo fenomeno (i diversi livelli, le successive astrazioni ...) solo a causa del successivo aumento di complessità di un sistema a noi perfettamente noto, e perfettamente riducibile "in linea di principio" agli elementi di base, *non c'è ragione* di ricorrere ad altre motivazioni per spiegare l'insorgere di situazioni analoghe anche in altri ambiti della scienza. Non c'entra la "causalità dall'alto"; non c'entrano le "proprietà emergenti"...

A me sembra che questa sia una situazione assolutamente generale, per qualunque ambito di conoscenza: è il nostro modo di pensare, nella presente fase della nostra evoluzione non ne conosciamo altri. Possiamo studiare un sistema complesso nei minimi dettagli, oppure possiamo ricavarne strutture di cui esploriamo la funzionalità, e questo sappiamo e possiamo fare a livelli successivi. In molti casi i livelli si diversificano al punto che non sono neppure le stesse persone a occuparsene: per ciascun livello occorre un diverso tipo di studio e di formazione. E così nascono le varie specializzazioni, che poi a volte diventano scienze fra loro distinte.

Se ci fosse bisogno di un'altra conferma, basta guardare la storia della scienza negli ultimi secoli e notare come man mano si sia ridotto, fino ad arrivare oggi praticamente a zero, il numero di persone con conoscenze enciclopediche, universali.

In realtà, se volessi davvero affrontare il tema della genesi delle diverse scienze, non potrei certo ridurlo soltanto alla scaletta (nella versione di Popper o nella mia, *ad libitum*). È chiaro che entra in gioco anche un altro aspetto,

per così dire orizzontale, accanto a quello verticale rappresentato nelle scalette. Per es. la chimica si differenzia dalla fisica anche per la sterminata varietà di composti esistenti o sintetizzabili, di cui bisogna conoscere e classificare le proprietà. Ancor peggio succede nelle scienze della vita: l'estrema varietà (anche storica) del mondo biologico impone un altro tipo di conoscenze, di organizzazione concettuale... Tutto vero, ma non posso addentrarmici, e del resto non ne ho bisogno: non mi pare che la distinzione "orizzontale" imponga un diverso approccio al problema della riducibilità.

* * *

E con questo sono arrivato in fondo, nei limiti delle mie capacità, circa il tema che mi ero proposto. Va da sé che le mie sono ipotesi, punti di vista probabilmente discutibili da chissà quanti lati; se ho voluto proporveli è stato più che altro per mostrare che sul riduzionismo si può anche pensare in modi diversi, uscendo da certe schematizzazioni un po' semplicistiche che capita spesso di trovare.

Di che parleremo la prossima volta? Non lo sapete voi, ma non lo so neppure io... A presto.