Computer science 1

February 24, 2022

Contents

List of Figures

1 Introduzione.

L'informatica (computer science) è una vasta disciplina che studia

- Algoritmi
- Rappresentazione dati
- Informazione
- Teoria del calcolo, logica, matematica
- Ciò che può essere automatizzato
- Automatizzazione del calcolo
- Sistemi elettronici automatizzati
- Teoria e applicazione dei linguaggi di programmazione

I primi due sono gli aspetti essenziali.
L'informatica è fondamentale per tutte le scienze, grazie ad essa il metodo sperimentale è cambiato, ad esempio attraverso simulazioni di super computer.

Terminologia.

- Hardware: componenti fisici di una macchina
- Software: tutto ciò che non è tangibile
- Computer e calcolatore

2 Informazione.

Si accenna alla teoria dell'informazione. Essa è una disciplina dell'informatica e delle telecomunicazioni che si occupa di misurazione, trasmissione, analisi ed elaborazione di informazioni. Si definisce "bit" come l'unità fondamentale di informazione. La grandezza che misura la quantità di dati prende il nome di entropia ed è espressa come numero di bit necessari per immagazzinare o trasmettere l'informazione. Tutto questo venne introdotto nel 1948 da Shannon con "*A Mathematical Theory of Communication*".
Nel corso si utilizzano le parole bit e informazione in modo più riduttivo. Infatti, il bit è una quantità minima di informazione che può assumere solamente due stati. Si può pensare che il bit sia una cifra binaria (**B**inary dig**it**). Una sequenza di N bit può assumere 2^N diversi valori.

Sistemi numerici. Oltre il sistema numeri decimale a cui si è abituati, l'informatica è solita utilizzare i sistemi numerici binario e esadecimale.

Bit e byte. Una sequenza di 8 bit costituisce un byte che può assumere $2^8 = 256$ diversi valori. Ogni byte si può considerare due stringhe di 4 bit una accanto all'altra, così si può scrivere con solo due cifre esadecimali: 0001 1010 = 11010 = 1A. Per distinguere le basi si usa scrivere 0b11010 = 0x1A, oppure 11010B = 1AH = 26D, dove B per binary, H per hexadecimal, e D per decimal.
La rappresentazione in forma binaria risulta particolarmente facile data la bassa quantità di stati diversi che bisogna rappresentare. Infatti, i primi metodi di immagazzinazione di informazioni faceva uso di schede perforate (le cosiddette punch cards). In era contemporanea, gli stati dei bit sono immagazzinati in circuiti elettronici in cui lo zero o l'uno vengono distinti in base alla tensione sopra una certa soglia del circuito: sotto la soglia equivale a zero, sopra la soglia equivale ad uno.
I primi circuiti utilizzati ad immagazzinare informazioni a lungo termine sono stati le memoria a nuclei magnetici.

Esempio. Di seguito si ha un esempio dell'utilizzo dei bit per comunicare un messaggio. Si osserva il paracadute della backshell del rover Perseverance:

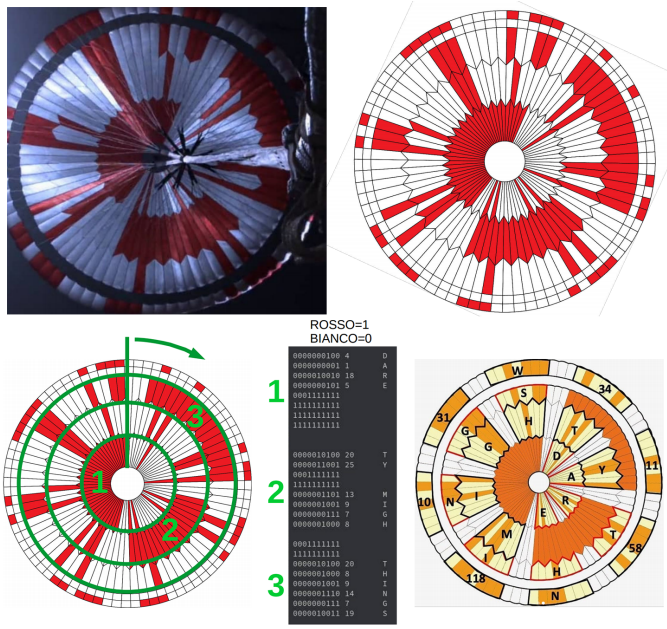

Figure 1: Paracadute e sua schematizzazione.

Il numero di bit esprime la dinamica con cui può essere rappresentata una granedzza fisica. Dato un insieme di M elementi diversi, si ha bisogno di una dinamica di N bit tali per cui $2^N \geq M$.

3 Elaborazione informazioni.

Le macro aree di cui si occupa l'informatica sono l'algoritmo, cioè la prescrizione astratta di calcolo utilizzata per risolvere un problema; la macchina, cioè l'hardware che eseguono i calcoli; il linguaggio, cioè il software che traduce l'algoritmo in un programma per la macchina.

Si individuano le informazioni di input da fornire all'algoritmo il quale fornisce le informazioni di output. Per tradurre l'algoritmo in linguaggio bisogna codificare ed organizzare in maniera astratta le informazioni di input. Quindi, bisogna scrivere un linguaggio e software capaci di tradurre l'algoritmo in istruzioni per la macchina, e capaci di codificare ed organizzare i dati di output. Infine, serve una macchina capace di ricevere le informazioni di input, processarle e poi "stampare" la rappresentazione dei dati di output.

3.1 Algoritmo.

Un algoritmo è una sequenza finita di istruzioni ben determinate ed eseguibili attraverso un procedimento meccanico, cioè un procedimento realizzabile da una macchina priva di libero arbitrio, che produce con certezza una soluzione per una classe di problemi.
Quindi deve essere:

- Finito: costituito da un numero finito di istruzioni, una sequenza che ha una fine;
- Deterministico: dai dati in input, si ottengono i medesimi risultati in output;
- Non ambiguo: indicazioni chiare, eseguibili nello stesso modo indipendentemente dall'esecutore;
- Generale: offre una soluzione per tutti i problemi della stessa classe.

Si dice che un problema è calcolabile quando è risolvibile mediante un algoritmo.

Dati. I dati sono rappresentazioni delle informazioni in ingresso (input), in uscita (output), e di qualsiasi informazione che viene temporaneamente immagazzinata durante l'esecuzione dell'algoritmo. Essi rappresentano informazioni relative a quantità su cui è possibile eseguire operazioni logico-matematiche. Sono di tipo diverso: il tipo è un attributo che specifica come interpretare, come codificare la rappresentazione dell'informazione e quali operazioni si possono eseguire. Infine, essi possono essere organizzati in strutture che ne definiscono le reciproche relazioni.

Esempio. Sia $f(N) = \sum_{i=1}^{N} i^2$. Si possono individuare i dati: N è un dato di input; i è una variabile che assume valori interi da 1 a N; i^2 è l'addendo; $SOMMA$, è il risultato dell'accumulazione dei valori di i^2. Si sceglie adeguatamente la grandezza massima che ciascun dato può assumere. Si legga $N = 7$, si pone $i = 1$ e la $SOMMA = 0$. Questi spazi sono stati allocati alla rappresentazione dei dati. Ogni cella allocata presenta un indirizzo. A questo punto dato che $i < N$, segue che $N = N$, $i = i + 1$, si calcola i^2 e infine $SOMMA = SOMMA + i^2$; e così via fin quando $i > N$. Quindi, si stampa la somma.

Costo computazionale. Il costo computazionale dipende dal numero di operazioni eseguite. A parità di dati in ingresso, un algoritmo costituito da un certo numero di istruzioni ha un costo computazionale minore rispetto ad un algoritmo con più istruzioni.

3.2 Programma.

Il termine programma può avere tre significati: il codice sorgente, il file eseguibile, il processo di esecuzione.
Il codice sorgente è un testo espresso in un linguaggio di programmazione. Il file eseguibile è la traduzione del codice sorgente nel linguaggio macchina. Il processo è un termine usato per il programma quando esso è in esecuzione dalla macchina.
I linguaggi si dividono in linguaggi formali e naturali. I linguaggi naturali sono parlati dalle persone a seguito di un'evoluzione storica. I linguaggi formali sono stati progettati per applicazioni specifiche; essi sono costituiti da simboli in un certo alfabeto (matematica, chimica, musica, ecc). Nei computer i linguaggi di programmazione sono usati per esprimere i calcoli. I linguaggi si dividono in basso e alto livello. Un linguaggio di basso livello differisce di poco dal linguaggio macchina e non c'è astrazione rispetto il funzionamento fisico del calcolatore. I linguaggi ad alto livello forniscono una grande astrazione dai dettagli del funzionamento della macchina. Per questo un linguaggio ad alto livello permette di non preoccuparsi della macchina specifica su cui il programma deve essere eseguito.
In questo corso si segue un tipo di programmazione imperativa, strutturata e procedurale (che differisce da quella orientata ad oggetti, come in C++ e Java).
Si riscrive l'esempio di prima in Python: Si mostra in C: Perché usare C al posto di Python? Posto $N = 70000000$, Python impiega 14 secondi, mentre C solamente 0.2 secondi. La differenza sta nel fatto che C è un linguaggio che dev'essere compilato, cioè tradotto a linguaggio macchina. Per Python c'è un interprete che traduce in tempo reale le istruzioni.

Programmazione strutturata. Il teorema di Böhm-Jacopini afferma che un linguaggio strutturato deve avere delle strutture di controllo: la sequenza, la selezione ed il ciclo (iterazione). La sequenza è il verso in cui un testo viene letto, cioè esiste un ordine in cui le operazioni vengono svolte. La selezione è la possibilità di scegliere in base ad una condizione quale strada percorrere. L'iterazione è la possibilità di ripetere le stesse operazioni fin tanto che una condizione non cambia il proprio valore. Inoltre, il teorema afferma il requisito di completezza, cioè tutte le strutture di controllo devono essere presenti; deve essere presente un punto di entrata e uscita; e infine devono essere componibili, cioè sono possibili l'annidamento e la ricorsione.

3.3 Macchina.

Von Neumann. Per macchina si intende architettura, nello specifico l'architettura di Von Neumann. Gli oggetti principali sono la *Central Processing Unit*, CPU, e *Random Access Memory*, RAM. Questi due oggetti sono collegati attraverso un canale detto BUS, il quale potrebbe servire anche altri dispositivi. Sia la CPU che la RAM contengono delle tabelle dette registri. Ogni registro ha due colonne, una per l'indirizzo e l'altra per il valore del dato. Ogni cella di memoria RAM presenta dati in 8-bit. I registri all'interno della CPU sono molto minori di quelli della RAM. I registri rappresentano delle istruzioni da svolgere e dei dati da maneggiare dalla CPU. I dati in memoria vengono elaborati da un'unità aritmetico-logica la quale esegue le operazioni aritmetiche e logiche. Inoltre, la CPU presenta una unità di controllo che supervisiona l'esecuzione delle istruzioni. Le operazioni sono scandite da un metronomo, detto *clock*. Esso serve a sincronizzare tutti i dispositivi del computer. Esistono dispositivi di *I/O*, cioè **I**nput/**O**utput, detti

anche *controller*, in quanto consentono alla CPU di leggere e scrivere dati su periferiche esterne le quali sono collegate alla CPU tramite un BUS. Il *clock* è un cristallo al quarzo che produce un'onda periodica. Inoltre, esiste la memoria di massa, *hard drive*, *disco fisso*, dove vengono immagazzinate i dati. Oltretutto, il programma in esecuzione ed i dati utilizzati vivono nella memoria RAM: non c'è un oggetto dedicato al programma in esecuzione e uno dedicato alla rappresentazione dei dati.

Memoria. Le memorie citate fin'ora sono di tipi diversi. I registri e la RAM sono memorie volatili, dall'altra parte sono presenti le memorie di massa. Inoltre, le prime due memorie dispongono di un'alta velocità di accesso.

Terminologia. Sono usati come sinonimi le parole: CPU, microprocessore, unità centrale, processore. Così come: memoria, memoria di lavoro, RAM. Pure: disco, memoria di massa, drive, disco fisso, disco rigido, hard disk, hard drive; sebbene hard disk indica una classe di memorie di massa. Inoltre, le parole chip e microchip, sebbene siano termini generici, in questo corso indicato il processore. Oltretutto, si utilizza il termine memoria cache, o semplicemente cache; essa è una memoria intermedia tra i registri e la RAM stessa. La parole memoria si rifà alla RAM stessa. Per dischi esterni s'intendono le memorie di massa come USB, dischi esterni USB, CD, DVD, floppy disk e nastri magnetici. Il termine drive indica un dispositivo che gestisce un generico supporto di massa.

I computer possono assumere forme diverse. Tuttavia, da un punto di vista funzionale, i concetti chiave sono quelli espressi per l'architettura di Von Neumann.

Personal Computer. Il primo computer da scrivania, *desktop*, chiamato Olivetti 101 venne prodotto da Olivetti nel 1965. Non è presente un monitor, ma è presente un rullo su cui vengono stampati i dati di output. Nei linguaggi di programmazione è rimasta il gergo *print*, per indicare al programma di stampare a schermo.
I computer hanno iniziato a diventare più veloci e leggeri dopo l'invenzione dei circuiti elettronici integrati, *chip*.

Microprocessore. Esso è un insieme di circuiti elettronici integrati. Il primo microprocessore fu l'Intel 4004 del 1971. Il numero di bit di registro di un processore, limita il numero di byte che una memoria può avere.
L'elettronica digitale è stata miniaturizzata, questo perché ciò significa velocizzarne le operazioni. Questo fatto ha vari motivi:

- La velocità di clock determina il numero di istruzioni al secondo. Ogni istruzione consuma energia.
- Transistor più piccoli hanno una capacità di gate inferiore.
- Transistor più piccoli si scaldano meno, perché richiedono meno corrente e quindi meno energia.
- Le distanze sono inferiori, quindi i campi elettrici si propagano tra le connessioni in un tempo inferiore.

- Ad alta frequenza, le dimensioni dei circuiti contano, perché la lunghezza d'onda è comparabile con le dimensioni del circuito.

Dagli anni '70 fino al giorno odierno, la densità di transistor è andata aumentando secondo la legge di Moore.

Un modo per accelerare la velocità è di utilizzare più unità di controllo ed unità aritmetico-logiche così da compiere operazioni in parallelo.
L'hard disk presenta dei dischi magneto-ottici su cui delle testine meccaniche scrivono e leggono i dati. Questo è più lento e più soggetto a fallire, rispetto a drive a stato solido. La differenza tra hard disk, HDD, ed un solid state drive, SSD, è che questo secondo presenta delle memorie flash al posto di dischi. Sono presenti anche memorie NVMe M.2.
La scheda madre può alloggiare la CPU, la RAM, le memorie di bassa, il BIOS, la batteria tampone ed il clock.

Codice macchina. Il compilatore collega il linguaggio di programmazione e linguaggio macchina. I compilatori sono diversi a seconda dei linguaggi. Possono esserci più compilatori per lo stesso linguaggio e che funzionano per diversi sistemi operativi. In questo corso si usa *Gnu Compiler Collection*, gcc, con cui si interagisce attraverso un terminale.

Terminologia. Per file si intende un archivio di dati immagazzinati in una memoria fissa. Il termine anglofono file viene utilizzato in italiano sebbene esistano traduzioni. Questo perché file indica dei dati archiviati in modo generico.

Utilizzando gcc, si compila un programma tramite il comando `gcc nome_file -o nome_file`. Il file sorgente viene tradotto in numeri esadecimali a quattro cifre.
Esiste un passaggio intermedio tra il codice sorgente ed il codice macchina. Esso è il codice Assembly, cioè un linguaggio a basso livello leggibile da un umano e dalla macchina. Il codice Assembly è la rappresentazione umana del codice macchina.

Processo. Il file eseguibile di per sè non è ancora eseguito. Per fare ciò, il file deve essere caricato in memoria. Il file sorgente/eseguibile diventa un processo.

La CPU esegue un ciclo continuo di tre operazioni:

- Fetch (recupero), si ottiene l'istruzione da eseguire dalla memoria.
- Decode (decodifica), si interpreta l'istruzione.
- Execute (esecuzione), si esegue l'istruzione.

Quando un programma è caricato in memoria, nel calcolatore si ha:

- La CPU viene informata della prima istruzione del programma tramite un contatore detto program counter.

- Il programma viene eseguito seguendo le istruzioni che sono caricate in memoria.
- Il processore esegue le istruzioni leggendole dalla memoria e tuilizza la stessa memoria per conservare dati e variabili del programma.

Sistema operativo. Tutto questo è gestito da un particolare programma che è sempre in esecuzione: il sistema operativo, OS. Esso è caricato per primo nella memoria del computer ed svolge delle mansioni fondamentali. Il sistema operativo gestisce: CPU, gestione dei processi; RAM, gestione della memoria di lavoro; memorie di massa, gestione del file system; I/O, gestione delle operazioni di I/O.
L'OS gestisce i processi tramite un codice detto *scheduler*. Esso si occupa di distribuire il tempo di elaborazione della CPU attraverso diversi programmi, dando l'impressione che più processi sinao in esecuzione simultaneamente.
L'OS gestisce la memoria di lavoro e le operazioni di I/O: quando un processo vuole accedere ad una risorsa, la deve richiedere al OS. Inoltre, è sempre l'OS che interagisce con l'hardware I/O.
L'OS crea un'astrazione (il file) organizzati in una struttura gerarchica file system. C'è una cartella madre, la root directory /. I percorsi possono essere assoluti e relativi, la cartella superiore si indica con `..`.

Boot. Quando il computer è spento, tutte le informazioni sono salvate sulle memorie di massa, incluso il sistema operativo. Quando si accende, bisogna prelevare il sistema operativo dal disco fisso e caricarlo in memoria. La fase che va dall'accensione al momento in cui il sistema operativo è completamente caricato si chiama boot.
Il firmware è a metà tra software e hardware che svolge specifiche istruzioni codificate in modo non volatile all'interno di un circuito elettronico (eventualmente riprogrammabile). Nei computer il firmware è il BIOS.
Sulle schede madri esiste un chip al cui interno è presente un BIOS che è il primo programma caricato in memoria. Il BIOS prende il bootloader dal disco e lo carica in memoria, così che il bootloader possa caricare il sistema operativo.

4 Dati.

Si codificano, rappresentano e organizzano i dati. Data un'informazione, si sceglie una rappresentazione astratta dei dati che corrisponde ad una rappresentazione logica e fisica di tale dato.

Esempio. L'informazione è l'età. Si sceglie di utilizzare una rappresentazione in C utilizzando una variabile del tipo `int` per cui vengono allocati quattro byte in memoria. Nel momento in cui si assegna un valore alla variabile, la memoria viene scritta, prima si era solamente allocata.

Dato un problema, bisogna identificare le variabili necessarie. Esse possiedono un nome (identificatore), un tipo ed un valore. Tale variabile occupa uno spazio nella memoria di lavoro. Il tipo della variabile determina lo spazio occupato in memoria ed anche le operazioni che si possono svolgere su tale dato. In C, prima di utilizzare una variabile bisogna dichiarare il tipo `tipo identificatore;`. In C, si presentano alcuni tipo primitivi:

- Interi (integer): `short`, `int`, `long`.
- Virgola mobile (floating point): `float`, `double`.
- Caratteri: `char`.

Si possono dichiarare più variabili sulla stessa riga, così da assegnar loro lo stesso tipo.
La forma generale dell'assegnamento di un valore ad una variabile è `identificare = espressione`. Il simbolo di uguale non ha lo stesso senso matematico. Esso ha verso in quanto assegna il valore a destra alla variabile nominata a sinistra. Inoltre, l'espressione viene valutata con i valori correnti delle variabili. Il risultato viene scritto nella cella di memoria occupata dalla variabile nominata a sinistra.

Esempio. Si ottiene come risultato 13.

Selezione. Se la condizione è valida, allora si eseguono certe istruzioni, altrimenti altre istruzioni. Una condizione può essere vera (1) o falsa (0). Alcuni esempi

- `x<y`
- `n>=0`
- `denom == 0`
- `denom != 0`

In C la selezione è:

Esempio.

Iterazione. Fino a che vale una condizione, si ripetono delle istruzioni.

Esempio.

4.1 Algoritmo.

Dato un logaritmo, bisogna dimostrare che la soluzione algoritmica proposta sia corretta e finita. Bisogna chiedere quale sia il costo computazionale, come indicarlo e come confrontare due algoritmi che risolvono lo stesso problema attraverso il loro costo.

Esempio. Si mostra l'esempio dell'ordinamento, in particolare la selezione ripetuta del minimo. Si impilano delle carte numerate e l'ultima è un cartoncino. Si prende la prima carta e la si confronta con la seconda. Se la prima è maggiore della seconda, allora questa viene posta alla fine del mazzo, mentre si prende la seconda e la si pone da parte. Ora, si confronta la terza carta del mazzo con quella messa da parte, e rimarrà da parte quella con valore minore. Si va avanti in tale maniera fino ad arrivare al cartoncino. Si ripete tale operazione con il resto delle carte, quindi ponendo la seconda carta più bassa accanto alla prima che è già da parte. Si ripete tutto questo fino a quando delle carte originali non rimanga solamente il cartoncino. In questo modo si saranno ordinate le carte in modo crescente.
Si analizza il costo computazionale. Per un mazzo di N carte, nel primo ciclo, si fanno

$N-1$ confronti nel peggiore dei casi (non si confronta il cartoncino). Nel secondo ciclo si fanno $N-2$ confronti nel peggiore dei casi. Quindi, sempre nel peggiore dei casi, il numero di confronti fatti è la somma dei primi $N-1$ interi, cioè $\sum_{i=i}^{N-1} i = \frac{N(N-1)}{2}$. Il costo computazionale è $O(N^2)$.
Si supponga di avere delle carte numerate. Si dividono in due gruppi e si applica l'algoritmo di ordinamento su ciascun mazzo. Poi si prende la prima carta dei due insiemi e si confrontano ponendo quella minore per prima in un nuovo mazzo e si ripone l'altra sul mazzo dov'è stata presa. Si ripete il procedimento fin quando si possono più fare confronti. Così si è ricomposto il mazzo originale in modo ordinato.
In questo caso il costo computazionale di ordinare i due mazzi è $2\sum_{i=1}^{\frac{N}{2}-1} i$. Per fondere i due mazzi si fanno al massimo $N-1$ confronti. A parità di numeri, questo algoritmo ha un costo minore di quello precedente.
Questa operazione di divisione e unione si può operare anche nel caso limite di considerare N mazzi da 1 carta ciascuna e poi unirli a due a due. Si supponga di avere $N = 2^x$ carte. Per 2^x mazzi si ha una sola carta per mazzo. Passando a due carte per mazzo, quindi 2^{x-1} mazzi, bisogna fare 1 confronto per mazzo, cioè 2^{x-1} confronti totali. Passando ad 2^n carte per mazzo, quindi 2^{x-n} mazzi, bisogna fare $2^n - 1$ confronti per mazzo, quindi i confronti totali sono $2^{x-n}(2^n - 1)$, tale numero è dello stesso ordine di infinito di $N = 2^x$. Inoltre, per ottenere il mazzo originale bisogna compiere $x = \log_2 N$ passaggi, una volta per smezzare, un'altra per unire. Quindi, l'algoritmo ha costo computazionale $O(N \log_2 N) = O(N \ln N)$.

4.2 Array.

I dati strutturati sono aggregazioni di dati organizzati o legati secondo una relazione. Il tipo più semplice è l'array (IT. vettore). Essi sono utili perché spesso i dati che interessa trattare sono organizzati in liste, vettori o tabelle. Quindi è utile avere un solo nome di variabile che indichi una successione di molti elementi. Gli array sono composti da elementi dello stesso tipo individuati da un indice che parte da 0 e sono distribuiti in modo consecutivo nella RAM.
In C si può scrivere un array come `tipo nome_array[dimensione]`.

Gli elementi di un array in C sono tutti dello stesso tipo. Si organizza un numero arbitrario di valore dando un nome unico che identifica i dati. A ciascun array viene assegnato un blocco di memoria della lunghezza adatta per contenerne tutti gli elementi. Gli array non contengono l'informazione della loro dimensione: bisogna, eventualmente, salvarla in un'altra variabile. Si accede ad un singolo elemento con il nome del vettore e l'indice dell'elemento.

4.2.1 Operazioni.

- `val[5] = x+3`
- `val[i+1] = val[i]*2`
- `temperature[0] = -3.4`
- `sigla[2] = 'a'`

- `if(val[i]<0)...`

Bisogna prestare attenzione quando si maneggiano degli array. Per dare dei valori agli elementi degli array bisogna utilizzare un ciclo for: Similmente, per stampare i valori bisogna usare un ciclo:

Esercizio. Scrivere un codice che copi i valori degli elementi di un vettore in un altro vettore, impostando un'iterazione nella quale si copia un elemento per volta nella posizione omologa.

Scambiare di posto i valori di due elementi.

Attraversare tutto l'array per eseguire un'operazione.

Quando viene dichiarare un array, viene allocato lo spazio in memoria per contenerlo. Tuttavia, le celle di memoria così assegnate potrebbero contenere informazioni preesistenti o essere in uno stato indefinito. Finché non si assegnano dei valori, non si sa quali valori si ottengono qualora si leggessero gli elementi dell'array.
Un modo di impostare i valori iniziali di un array è quello di inizializzarne tutti i valori a zero 0: `int myArray[10]={0}`; oppure specificando i singoli valori da inizializzare: `int myArray[]={12,34,1,0,34,1}`, in questo caso si può omettere la lunghezza in quanto è evinta dagli elementi assegnati.

Si possono costruire array multidimensionali i quali necessitano più indici. Ad esempio: `int valori[5][3]` quest'array ha 5 righe e 3 colonne. In memoria i valori vengono assegnati fissando il primo indice e scorrendo il secondo.

4.3 Funzione.

La funzione è una sezione di un programma che svolge un compito specifico, riceve degli argomenti in ingresso e produce un risultato od un effetto. Quando si scrive del codice si vorrebbe utilizzare nuovamente alcune idee, algoritmi, operazioni, specificando di volta in volta dati di ingresso diversi, e intercettarne l'uscita per svolgere altre operazioni successive. Oppure si vorrebbero ripetere alcune procedure più volte, in punti diversi del codice. In fase di progettazione, si vorrebbero scrivere quali sono le funzioni di parti del codice e come possono "chiamarsi" l'una con l'altra, ragionando successivamente su come realizzarle in pratica. Questo è un modo di ragionare detto top-down: si passa dall'astratto al concreto, descrivendo una visione generale dei problemi per affrontare e rifinire i dettagli in seguito.
Una funzione ha dei parametri di input detti argomenti; un corpo, cioè la porzione di codice che ne definisce le operazioni; un tipo, che definisce il tipo di dati che vengono restituiti. Se una funzione ha un'istruzione che restituisce dati in output si chiama funzione, se invece svolge operazioni senza restituire dati, si dice procedura e di solito ha il tipo void.
Un programma in C consiste di una serie di definizioni di funzioni. Il nome `main` denota una funzione speciale, che dev'essere presente in ogni programma e dalla quale prende avvio l'esecuzione. Si cerca di mantenere il `main()` il più semplice possibile, come se fosse una sorta di sommario delle azioni da svolgere, e spostare i dettagli in funzioni esterne: si articola il codice in funzioni.
Da una funzione è possibile chiamare o invocare altre funzioni, specificando i valori degli

argomenti; al termine dell'esecuzione della funzione chiamata, la funzione chiamante riceve il valore prodotto dalla chiamata.

Le due parti principali sono l'intestazione ed il corpo. In questo è presente una funzione speciale, `return`, che segnala la fine dell'esecuzione e specifica il valore prodotto dalla funzione: I parametri formali sono un elenco di dichiarazioni di variabili. Sono formali nel senso che quando si intesta la funzione, le variabili non esistono ancora in memoria.

Gli identificatori degli argomenti nell'intestazione di una funzione sono chiamati argomenti formali. I valori specificati nella chiamata di una funzione chiamati argomenti reali. Un argomento reale può essere una variabile od un argomento specifico. Si dice che la funzione restituisce un valore alla funzione chiamante. Le variabili utilizzate da una funzione vivono in uno spazio di memoria che viene creato ogni volta che si chiama la funzione e che viene cancellato quando quella chiamata termina. Tale spazio di memoria si chiama record di attivazione di una funzione. Le chiamate delle funzioni possono essere composte.

Gli argomenti delle funzioni visti finora vengono copiati nei parametri reali e quindi le funzioni lavorano su delle copie dei valori. Questo resta vero per i singoli elementi di un vettore. Tuttavia, con gli array succede una cosa diversa. Bisogna immaginare che l'identificatore di un array sia l'indirizzo in memoria della posizione dell'array.

La funzione `raddoppia` è void, infatti non restituisce alcun valore, ma modifica direttamente l'array, non crea una copia come per le singole variabili o valori.
Inoltre, il nome del parametro formale è diverso dal nome del parametro reale.

Esempio. Il nuovo nome che identifica l'array, `a`, punta allo stesso indirizzo di `x`, pertanto modificando gli elementi di `a` si stanno modificando gli elementi di `x`, perché essi puntano alle stesse celle di memoria. Finita la funzione, le celle in memoria occupate dalle variabili locali vengono liberate.
Quando si passano array come argomenti si possono modificare i dati nell'originale e non in una copia. Questo perché si passa una copia dell'indirizzo di memoria in cui è memorizzato l'array. Quindi si accede ancora alla memoria in cui è presente l'array originale. Si possono passare anche array multidimensionali come argomenti di funzioni, ma tutte le dimensioni tranne la prima devono essere specificate. Non si può restituire un array nel `return` di una funzione, perché farebbe riferimento a un'area di memoria che si trovava nel record di attivazione ormai eliminato.

4.4 Campo di visibilità.

La visibilità (scope) è l'esistenza e la possibilità di richiamare un identificatore in un determinato punto del programma. Una variabile non "esiste" sempre e non è accessibile da ogni parte del codice, ma solo nella sezione del programma detta *scope* della variabile, dove essa è nota. Esistono, quindi, regole che dicono quale "pezzo di codice" "vede" una certa variabile. Una variabile è "visibile" nel blocco di codice che la contiene, in pratica dal punto in cui è definita alla chiusura della parentesi graffa che contiene la definizione. Nel `for` le variabili definite nell'inizializzazione sono visibili solo nel corpo del `for`. Gli identificatori dei parametri di una funzione sono definiti solo nel corpo della funzione, e così le variabili dichiarate all'interno della funzione stessa.

Anche i nomi delle stesse funzioni sono "visibili" solo dopo la dichiarazione (prototipo) della funzione stessa. Si possono dichiarare variabili globali, al di fuori del corpo delle funzione, visibili in tutte le funzioni che seguono la loro dichiarazione nel codice sorgente. Il loro uso va limitato e in generale si deve evitare di ricorrervi.

Si consideri: Si supponga di chiamarla in un frammento di programma: Da ciò si ha `v=10` e `w=5`.

Il modo in cui si è visto come si passano i parametri nelle funzioni per variabili semplici (non array) si dice che è un passaggio di parametri per valore: si creano delle copie nella memoria del computer dei valori che vengono passati alla funzione. Esistono anche il tipo di passaggio per riferimento (tipico degli array): non si crea una copia, ma l'indirizzo; l'unica rappresentazione in memoria che esiste è il riferimento.

4.5 Strutture.

Si supponga di osservare il firmamento. Ci si focalizza sulla costellazione dell'orsa maggiore. Ad ogni stella si associa un nome, una nomenclatura di Bayer, una magnitudine apparente ed una distanza. L'insieme di queste informazioni si può intendere come la costellazione.
In C si costruisce tale tabella. Si utilizza la struttura `struct`: Alla prima riga si intesta la tabella. Alla riga otto si specifica che la quantità di elementi in `UrsaMaior` è 7 ed ogni elemento è di tipo `struct Stella`.
Da subito bisogna stabilire il tipo di dato in maniera univoca. La definizione della struttura è terminata dal punto e virgola.

Gli algoritmi lavorano con dati strutturati. Tali strutture di dati possono contenere informazioni eterogenee, ciascuna identificata da un campo, ognuno con un suo nome e tipo.
Strutture, struct e record sono sinonimi che indicano aggregati di dati; ciascun singolo membro (cioè la variabile) di una struttura è detto campo (field); ciascun dato strutturato (cioè il valore particolare di una variabile) è detto record.
È possibile definire vettori di record. Un campo può essere a sua volta un record, o un vettore, o altro. In questo modo è possibile comporre strutture via via più complesse.
Si vede la struttura: La struttura va definita fuori da ogni funzione e prima di dove viene utilizzata. Ogni campo ha un suo tipo. I tipi possono essere diversi tra loro e possono essere altre strutture o array. Inoltre, bisogna terminare la struttura con il punto e virgola perché si sta definendo un modo di raggruppare dei dati.
Si può anche dichiarare una struttura come: In questo modo si dichiara che `sp` e `pt` sono del tipo `struct CoordinateTemperatura`.

Si vedono alcuni esempi. Di seguito si vede uno dei modi per assegnare dei valori al campo. Dichiarando la variabile `p`, il programma alloca lo spazio nella memoria, senza inizializzarla. Quando si assegna un valore ad un campo della variabile, la memoria viene riempita con i valori assegnati. Dichiarando due variabili diverse, ma con la stessa struttura, non si utilizzano le stesse celle di memoria.
Assegnare una struttura ad un'altra struttura ne copia tutti i campi. Si si passa una struttura come argomento a una funzione, essa viene copiata (contrario agli array e simile alle variabili), quindi l'originale non viene modificato. Non è possibile confrontare due strutture con l'operatore di uguaglianza, come avviene invece con le

normali variabili non strutturate.

Esiste la funzione `typedef` per definire il nome di un nuovo tipo di dato. Lo si può usare per evitare di scrivere `struct NomeStruttura` nel seguente modo: `typedef struct NomeStruttura Abbreviazione;`. Questo consente di usare "`Abbreviazione`" al posto di `struct NomeStruttura`: In questo modo si restituiscono più valori da una funzione (cioè i campi di `c`).

5 Rappresentazione dei dati.

Un generico numero a si può scrivere come

$$a = \sum_{i=-N}^{M-1} c_i b^i$$

con coefficienti c_i e di base b, con N numeri prima della virgola e M dopo.

5.1 Numeri interi.

Si supponga di voler rappresentare un numero con il segno. Con M bit si possono rappresentare 2^M numeri. Si può immaginare di traslare lo zero: questa è la rappresentazione in eccesso a N, si rappresentano i numeri che vanno da $-N$ fino a $2^N - (N+1)$. Di solito si pone lo zero nella casella a destra del centro (il centro non è una casella, perché il numero di caselle è pari, quindi non c'è centro).

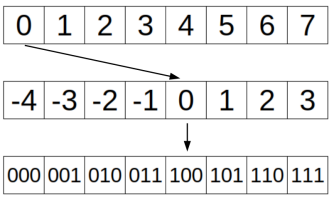

Figure 2: Rappresentazione in eccesso.

In generale, si dice complemento k di un numero m in base b rappresentato con n cifre, il numero $m + k = b^n$. Infatti:

$$m + k = b^n \iff k = b^n - m = b^n - 1 + 1 - m$$

Il numero $b^n - 1$ è formato da una sequenza di cifre massime, quindi per ottenere k basta trovare, dato m, il complemento di ciascuna cifra rispetto alla cifra massima, poi sommare 1.

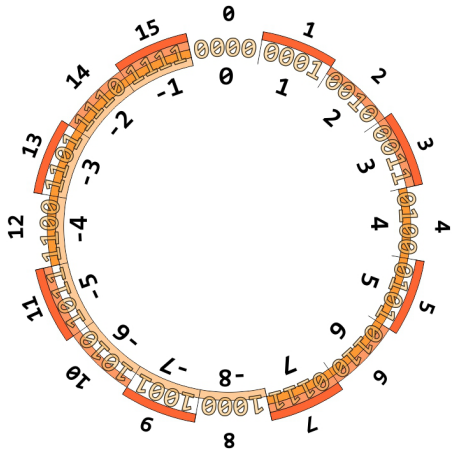

Figure 3: Rappresentazione in complemento.

Tutti i numeri la cui cifra più significativa è 1 sono numeri negativi.

Esempio. In base 10 (dove la cifra massima è 9), il complemento a 10 del numero 134 espresso con 4 cifre è: $9865 + 1 = 9866$.
Per la base 2 il complemento di 0 è 1 e il complemento di 1 è 0: per passare da 5 a -5 si fa 0101 diventa $1010 + 1 = 1011$.
Quindi, per fare $5 - 2$ si può fare $5 + (-2)$ con quattro bit. Si somma la rappresentazione in colonna dei numeri, e si troncano a sinistra i bit oltre quelli prefissati: $0101 + 1110 = 10011 \rightsquigarrow 0011$ che è 3.
In C la funzione `sizeof(tipo)` serve per ottenere l'effettiva dimensione in byte che può variare a seconda degli standard.

5.2 Numeri reali.

I numeri reali non sono rappresentati esattamente all'interno di un computer, essendo il loro insieme denso, mentre le possibilità di un calcolatore sono solo di immagazzinare informazioni discrete. Tuttavia esistono metodi e convenzioni per rappresentarli in modo approssimato, tenendo eventualmente conto delle differenze tra la rappresentazione ed il numero vero. Si noti che non si tratta solo di un problema di numero di cifre.
Per i numeri reali si adotta una convenzione (definita nello standard IEEE 754) che definisce come sono rappresentati i numeri floating point (in virgola mobile) e come funzionano le operazioni su di essi. Il numero totale di bit può cambiare a seconda della precisione desiderata: singola precisione, 32 bit; doppia precisione, 64 bit. I bit a disposizione vengono utilizzati per tre informazioni diverse: segno, esponente e mantissa. Inoltre, vengono assegnate delle rappresentazioni per risultati impossibili da calcolare dette NaN e per risultati "infiniti", $\pm$Inf.
Il numero da rappresentare si calcola quindi come $\pm \text{MANTISSA} \cdot 2^{\text{ESPONENTE}}$. Si converte il numero con la virgola da base 10 in base 2. Si scrive il numero in base 2 spostando la virgola in modo da avere solo una cifra unitaria alla sua sinistra e scegliendo di conseguenza l'esponente da dare a 2. Si assegna un bit per il segno. Si usano 8 o 11 bit per rappresentare l'esponente in eccesso a 127. Per la mantissa si usano 23 o 52 bit. Viene immagazzinata allineandola a sinistra omettendo il primo 1. Se l'esponente è

composto da soli 0, allora non si sottintende l'omissione dell'1 iniziale (forma denormalizzata).
Per convertire un numero decimale con parte intera e parte frazionaria, si converte la parte intera come è noto. Dopo la virgola si scrive la conversione della parte frazionaria. Per convertirla: si moltiplica la parte frazionaria del numero dato per 2; si continua a moltiplicare il risultato ottenuto per 2 tenendo presente che, se il numero ottenuto è maggiore di 1, si sottrae 1; si prosegue fino a quando non si ottiene o un risultato uguale a uno oppure fino a quando si ottiene un risultato già ottenuto in precedenza. Quando si ottiene un risultato già ottenuto in precedenza significa che il numero binario è periodico. Ciò può accadere anche se il numero decimale di partenza non è periodico. Il numero binario cercato è dato dalle parti intere dei prodotti ottenuti partendo dal primo.

Esempio. Per convertire 17.15_{10} nel formato floating point: si converte 17_{10} in binario 10001_2; si converte 0.15_{10} in binario $0.00(1001)$. Quindi $17.15_{10} = 10001.001001100110011...{}_2$; si sposta la virgola di 4 posti (quindi l'esponente è 4) per ottenere $M = m - 1 = 00010010011001...$. Per l'esponente $E = e + 127 = 131_{10} = 10000011_2$. Il numero è positivo, quindi $S = 0$. Pertanto, $17.15_{10} = 0\ 10000011\ 00010010011001100110011_2$.

Data una base $\beta \geq 2$ intera, un numero t di cifre significative, un intervallo $[L, U]$ di esponenti e interi, allora l'insieme dei numeri rappresentabili in virgola mobile in forma normale è:

$$\mathbb{F}(\beta, t, L, U) = \{0\} \cup \{x \in \mathbb{R} \mid x = (-1)^s \beta^e \sum_{i=1}^{t} d_i \beta^{-i}\}$$

Non si considera la forma denormalizzata. Inoltre, questo è un insieme finito che può solo approssimare i valori reali.
Tali valori hanno la stessa distribuzione sull'asse reale. Il numero di possibili mantisse diverse prescinde dal valore dell'esponente. Questo comporta che la distribuzione dei numeri rappresentabili non è omogenea: vicino allo zero la densità è maggiore.
Le frazioni che non sono potenze del 2 non sono rappresentabili esattamente.
L'errore che si compie è costante in senso relativo: la precisione di macchina stima tale errore. Data una base, B, e le cifre significative della mantissa, t, è:

$$E_{\text{macchina}} = B^{1-t}$$

Questo è l'epsilon di macchina ed è un indice del massimo errore relativo nell'approssimazione di un numero diverso da zero:

$$\left| \frac{\hat{x} - x}{x} \right| \leq E_{\text{macchina}}$$

Dove x è il numero da voler rappresentare e $\hat{x}$ è l'approssimazione.
In una rappresentazione a virgola mobile, i numeri rappresentabili esattamente sono più "densi" intorno allo zero, e si diradano man mano che aumenta il valore assoluto. In questo modo l'errore relativo è limitato, indipendentemente dalla grandezza del numero.

A causa della non esattezza delle rappresentazioni di numeri reali non è detto che le operazioni siano associative o commutative. Operazioni matematicamente equivalente possono dare risultati diversi a causa delle diverse approssimazioni. A volte è

consigliabile controllare non l'uguaglianza, ma guardare che siano sufficientemente vicini al valore corretto. In una sequenza di operazioni su numeri in virgola mobile, gli errori possono amplificarsi. la disciplina che si occupa dei problemi legati ai calcoli approssimati si chiama Analisi Numerica.

5.3 Caratteri.

Ogni carattere è rappresentato da un numero intero non negativo. Secondo la convezione ASCII, un carattere occupa 8 bit. Le cifre non corrispondono ai valori numerici. Ci sono caratteri speciali come il ritorno a capo (line feed). Nel codice Unicode si hanno dei caratteri a due byte per rappresentare i caratteri di tutti i vari alfabeti.
In C si può non specificare la lunghezza dell'array di caratteri. Inoltre, il C aggiunge un elemento finale `\0` che termina la stringa ed è un byte di soli zeri.

6 Indirizzi di RAM.

Riprendendo l'esempio di prima, quando si dichiara in questo modo l'array, la macchina associa ad un indirizzo di memoria un valore che corrisponde a ciò che si vuole rappresentare: in questo caso un valore intero che convertito tramite la tabella ASCII fornisce un carattere.
Il C permette di gestire gli indirizzi tramite i puntatori. Il valore di puntatore è l'indirizzo in memoria. In C si dichiara un puntatore nel seguente modo: L'asterisco indica che il tipo della variabile è puntatore a "tipo puntato".
Per usare i puntatori si deve:

- Scrivere o leggere un indirizzo di memoria nel puntatore.
- Scrivere o leggere il valore puntato dal puntatore.

6.1 Puntatori.

Si supponga di avere la seguente variabile `int a = 5;`. Per ottenere l'indirizzo in memoria si utilizza `&a`. Si vede cosa accade nella memoria: Al puntatore `c` viene assegnato l'indirizzo di memoria dove si trova `b`.
Si può assegnare ad una variabile di tipo puntatore tutto ciò che ha una posizione ben definita in memoria.
Per accedere al valore contenuto nella locazione di memoria puntata si esegue una dereferenziazione, ovvero si "segue" il puntatore. Mettere un asterisco prima del puntatore significa che bisogna prendere il valore puntato. Pertanto, in C il simbolo `*` è utilizzato per la moltiplicazione, dichiarazione di un puntatore, e come operatore di risoluzione di un puntatore.

Esempio. Uno degli usi per cui i puntatori in C sono fondamentali è il loro passaggio come argomenti a una funzione. Quando vengono passati ad una funzione, il puntatore viene copiato, cioè viene copiato l'indirizzo a cui esso punta. Questo significa che la funzione chiamata fa riferimento alla stessa locazione in memoria della funzione chiamante. In questo modo si passano gli argomenti "per riferimento" (by reference), invece che "per valore" (by value).

Esempio. Sembra chiaro a questo punto che i puntatori hanno molto in comune con gli array, e che gli array sono trattabili in molti casi, non tutti, come puntatori. Si è già visto, infatti, che non è possibile restituire un array attraverso il return di una funzione. Tuttavia, anche se le funzioni non possono restituire array, esse possono restituire puntatori.
Si possono usare puntatori a strutture, tuttavia per accedere ai singoli campi si deve dereferenziare il puntatore: Bisogna scrivere `(*p).x` ed esso si può scrivere anche come `p -> x`.

Esempio. Inoltre, è importante inizializzare il puntatore prima di compiere delle operazioni.

Il puntatore è una variabile il cui valore è un indirizzo di memoria. In un sistema il cui processore utilizza 32 bit per rappresentare un indirizzo di memoria nei suoi registri, la RAM può avere fino a 2^{32} locazioni di memoria indirizzabili, ciascuna grande un byte; in altre parola la dimensione massima delle RAM è 2^{32} byte, cioè circa 4 GB. In un tale sistema, per un puntatore è necessario e sufficiente usare un numero intero a 32 bit.
Gli indirizzi sono spesso rappresentanti in esadecimale. Tuttavia, bisogna indicare il tipo del puntatore perché si è definita l'aritmetica dei puntatori con i numeri interi.
Il tipo del puntatore serve a definire cosa succede quando si sommano o sottraggono valori interi da una variabile di tipo puntatore. Sommare un valore intero N a un puntatore significa spostarsi in avanti N locazioni di memoria, ciascuna di dimensione necessaria a rappresentare il tipo della variabile puntatore stessa. Se per esempio p è un puntatore di tipo double, che può puntare ad una variabile double rappresentata da 8 byte, sommare un valore intero N significa spostarsi di N locazioni di memoria di dimensione necessaria a rappresentare un double, ovvero di spostarsi di $8N$ byte. Sottrarre un numero intero N ad un puntatore significa invece, analogamente, spostarsi indietro di N locazioni di dimensione necessaria a rappresentare il tipo del puntatore.
Inoltre Manca la `&` in `int * p = &a` perché un array è rappresentato da un indirizzo di memoria: i nomi di array non sono variabili, ma puntatori. Pertanto, `p` si può utilizzare a tutti gli effetti come fosse un array. Infatti, `p+1` equivale a `&p[1]`, pertanto `*(p+1)` equivale a `p[1]`.

Esempio. Negli esempi visti, il numero di dati era fissato e noto a priori. Naturalmente, non è sempre questo il caso, e allora per situazioni più complesse bisogna capire come è divisa e utilizzabile in modo dinamico la memoria di un programma.

6.2 Stack.

In generale, per stack (in italiano, pila), si intende una struttura di danti di "impilata", di cui è accessibile solo un elemento, il primo della pila (per esempio un mazzo di carte dal quale non si può estrarre delle carte se non partendo sempre dalla prima in alto). Quando si parla di stack con riferimento ad un tipo di gestione della memoria, allora si chiama stack quel tipo (o area) di memoria usata in modo consecutivo e che si è usata finora. Ogni chiamata di funzione alloca un record di attivazione che contiene i parametri, le variabili locali, la cella in cui restituire l'uscita di una funzione. I record di attivazione sono aggiunti e rimossi come una "pila di piatti". Lo stack di attivazione del main è il più basso. Quando si chiama una funzione dal main, lo stack di attivazione di

tale funzione è posto sopra il main e non scompare fin quando la funzione non termina. Le variabili locali cessano di esistere quando la funzione termina. Si possono modificare tramite i puntatori solo i valori allocati da funzioni più "in profondità" nello stack. In generale, non si ha controllo su quando la memoria viene allocata e deallocata. D'altra parte questo è un vantaggio se non ne ci si deve occupare.
Si può gestire la memoria heap in modo che questa non sia necessariamente impilata e consecutiva, come lo stack. In questo caso, si deve richiedere esplicitamente l'allocazione della memoria. Si può decidere durante l'esecuzione quanta memoria allocare. Ma bisogna occuparsi di restituire la memoria una volta che non è più necessaria.

6.3 Malloc.

Si utilizza una funzione che si occupa di richiedere la memoria: malloc. Bisogna indicare la dimensione: per farlo bisogna dire quanti dati di un certo tipo deve contenere (si usa `N*sizeof(tipo)`). La funzione malloc restituisce un puntatore alla memoria allocata, cioè al primo byte della sequenza allocata. Una volta che non serve più la memoria, la si libera chiamando free.

Esempio. Se si volesse rappresentare N interi si scrive `int * a = malloc(10*sizeof(int))`, dove `a` diventa a tutti gli effetti in array, e si accedono agli elementi con `a[5]` oppure `*(a+5)`.

A volte si può voler fare un cast per forzare il puntatore ad essere del tipo corretto: `int * a = (int *)malloc(sizeof(int))`.
Se l'allocazione di memoria fallisce, viene ritornato il valore `NULL`, in quel caso la memoria non è stata allocata:

Esempio. Il record di attivazione di `crea_array` scompare dopo che le funzione restituisce il valore del puntatore che punta ad un array creato in memoria heap.

Al termine dell'esecuzione è buona norma deallocare la memoria che non si utilizza più.
Bisogna ricordarsi di allocare la quantità giusto di memoria, usando `sizeof` con il tipo corretto. Se si perde un puntatore ad un'area di memoria allocata con malloc quella memoria non verrà liberata: si ha un memory leak. Bisogna ricordarsi se la memoria è gestita in automatico (stack) o se invece la si è richiesta con malloc e quindi rimane finché non la si dealloca (heap).

Si possono anche dichiarare puntatori a puntatori.

6.4 Notazione polacca inversa.

Con la funzione pop si rimuove un elemento dalla cima dello stack, viceversa con al funzione push.
Si supponga di avere un'espressione $(9 \times 3) + (5/4)$. Si scrivono prima gli operandi e poi gli operatori: $(9\,3\times)(5\,4/)+$. Si possono così eliminare le parentesi: $9\,3 \times 5\,4/+$. Questo si può fare una volta che si definisce l'arietà di un operatore, cioè il numero di operandi che tale operatore richiede. Questo tipo di notazione è detto notazione polacca inverse.

Si utilizza lo stack. Se il valore letto è un numero, allora lo si inserisce nello tack. Altrimenti, se è un operatore, si ottengono i due operandi `x = pop()`, `y = pop()`, e si calcola il risultato `z = y op x` e lo si inserisce nello stack. Finita la lettura e l'algoritmo, il risultato è in cima allo stack.
Questo sottolinea come la rappresentazione dei dati sia più importante dello specifico algoritmo.

7 Strutture dinamiche.

In una pila (stack) l'ultimo elemento che si aggiunge (tramite push) è anche il primo a essere tolto (tramite pop). Questo tipo di struttura di dati si dice "last in, first out", LIFO.
Esistono organizzazioni di dati dette code (queue) in cui il primo a entrare è il primo a uscire: tale struttura è detta "first in, first out", FIFO.

7.1 Liste concatenate.

In questi casi il numero di elementi può cambiare. Tuttavia, si vorebbero strutture di dati più complesse che crescono e decrescono in base ai dati, in modo da avere anche buone prestazione ed un uso efficiente della memoria di un calcolatore.
Un modo di struttura è la lista concatenata: ogni elemento della catena è detto nodo e presenta tre dati. Si ha l'indirizzo del nodo, il valore del nodo, e l'indirizzo del nodo successivo. Pertanto, si possono concatenare i nodi. L'ultimo nodo punterà all'indirizzo `NULL`.
La lista è composta da un nodo di testa, un numero non specificato di nodi, ed un nodo di coda. In C si scrive una struttura del tipo: Il puntatore punta allo stesso tipo di struttura in cui si trova.

Con le liste concatenate si può:

- Creare una lista vuota.
- Inserire un elemento in qualsiasi posizione.
- Ricercare un elemento nella lista.
- Rimuovere un elemento.

Per ottenere una lista concatenata con zero elementi si definisce un puntatore `struct nodo *testa = NULL`.

Per aggiungere un elemento in testa alla lista si crea un nuovo nodo che punta al valore da aggiungere; il puntatore del valore aggiunto va a puntare al valore puntato dalla testa, la quale, ora, viene fatta puntare al valore aggiunto.
Il nome testa, che potrebbe essere l'identificatore di un puntatore, qui è l'unico nome necessario; i nodi successivi sono raggiungibili attraverso la catena di puntatori e indirizzi, ma non hanno necessariamente un "nome".
Un esempio di codice: Nel chiamante si ha, per esempio:

`testa = inserisci(26, testa);`.

Per eliminare un nodo bisogna assegnare al puntatore di testa l'indirizzo puntato dal nodo da eliminare; così poi da liberare il nodo da eliminare. Per esempio:

Si possono anche creare liste concatenate doppie: ad ogni nodo si aggiunge anche l'indirizzo del nodo precedente. Similmente, si può creare un buffer circolare, dove l'ultimo elemento punta al primo.

8 Alberi binari di ricerca.

Si è visto, con la notazione polacca inversa, come la notazione rappresentazione dei dati possa influenzare gli algoritmi, e vale anche il viceversa.
La scelta di un algoritmo è dettata dal costo computazionale.

Per esempio, si supponga di avere una lista di valori numerici interi nella quale cercare un numero. Non si sa cosa contiene, quindi bisogna scorrerla per trovarla. Ci si chiede quale sia il modo migliore per farlo. Si prova con l'algoritmo illustrato prima, scorrendo la lista dal primo elemento verso destra. L'algoritmo funziona, ma potrebbe non essere efficiente.
Supponendo il vettore ordinato, si parte dalla metà. Se il valore non è quello cercato:

- Ci si sposta nella metà di destra se il valore è maggiore del primo scoperto.
- Altrimenti a sinistra.
- Si ripete la procedura sul sotto-vettore di scelto.

Un aspetto importante è che l'algoritmo viene iterato.
L'algoritmo di ricerca binaria dà vita ad un'organizzazioni dei dati che facilità l'applicazione dell'algoritmo stesso.
Questa struttura astratta può essere utilizzata in modo diverso in un codice o in una strutturazione dei dati all'interno del calcolatore. Un modo di definire un vettore potrebbe essere nell'ordine in cui l'algoritmo controlla il valore centrale. Una semplificazione arriva dalla possibilità di utilizzare l'albero binario di ricerca. Esso è la rappresentazione naturale dell'algoritmo di ricerca binaria. Al valore centrale in ciascuna iterazione si associano due puntatori: uno di destra ed uno di sinistra. Se l'elemento puntato assente, allora al puntatore si assegna il valore `NULL`. Il nodo di partenza è la radice (root), i successivi sono sottoalberi. Ogni nodo a parte la radice ha un genitore (parent) ed un figlio detto foglia (leaf). Per inserire un nuovo nodo, si può effettuare la ricerca del suo valore, e laddove si incontra il puntatore `NULL` si inserisce il nuovo nodo.
In questo caso l'algoritmo ha forgiato la rappresentazione dei dati.

Si studia il costo computazionale dell'algoritmo di ricerca binaria. Ad ogni passo si elimina circa la metà degli elementi. Il numero massimo di confronti è $\log_2 N$, dove N è il numero di elementi. Se si raddoppia N, allora il numero di confronti aumenta di 1. Nella ricerca in cui si percorrono semplicemente il vettore ordinato in cerca del valore, nel peggior caso si compiono N confronti, e se raddoppia N raddoppia il numero di confronti. I due algoritmi hanno rispettivamente $O(\ln N)$ e $O(N)$.

9 Ricorsione.

Finora si sono viste le funzione che possono chiamare altre funzioni, anche ripetutamente se necessario. Esisstono casi in cui sarebbe utile esprimere la chiamata di una funzione a sé stessa. Si tratta di un ragionamento per ricorsione e si parla, quindi, di algoritmi o funzioni o metodi ricorsivi.
Nota terminologica: "ricorsione" ed "iterazione", o se si vuole "ricorsivo" e "iterativo", hanno significati diversi.

Esempio. Si consideri il fattoriale di un numero $n \in \mathbb{N}^+$:

$$n! = \begin{cases} n(n-1)! & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

È soprattutto uno strumento concettuale, che aiuta a trovare soluzioni algoritmiche. Non è detto che una funzione ricorsiva sia più efficiente di una scritta in ofrma non ricorsiva. Si può definire una cosa in termini di se stessa, è molto simile al ragionamento induttivo e alle dimostrazioni per induzione viste in matematica. Si dimostra un caso particolare, si dimostra l'implicazione della validità del caso successivo e così si passa dal particolare al generale.

Esempio. Bisogna notare che ogni volta che si chiama la funzione, si crea un nuovo record di attivazione, dove si copiano le variabili.
Si ha un caso base di cui si conosce il valore della funzione. Ogni chiamata genera un nuovo record di attivazione con una sua copia delle variabili locali e dei parametri. NOn si continua a chiamare la funzione all'infinito: si ha un passo ricorsivo in cui si chiama di nuovo la funzione fino ad un caso base.
Il caso è del tipo: Il caso base è un caso a cui si deve sempre ricondursi dopo un numero finito di chiamate. Quando si arriva al caso base, bisogna assumere di conoscere (o saper calcolare direttamente) il valore della funzione. Nella chiamata ricorsiva si riconduce il problema di trovare il valore della funzione in termini della funzione stessa.
Se non si arriva al caso base (oppure si richiede troppa memoria) si può avere uno stack overflow: si è riempito tutto lo spazio disponibile per lo stack.

Esempio. Alcuni esempi di funzioni ricorsive. La sequenza di fibonacci:

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n > 2 \\ 1 & \text{se } n = 0 \vee n = 1 \end{cases}$$

Un altro esempio può essere la lunghezza di una lista.

Si è già detto che un linguaggio strutturato deve avere delle strutture di controllo: la sequenza, la selezione ed il ciclo (iterazione); non si è parlato di ricosione, tuttavia si può considerare come estensione del caso di iterazione. Dunque, dato che un linguaggio strutturato non possiede tale esigenza, significa che si può sempre, con tali strutture di controllo, scrivere un programma.
La ricorsione e l'iterazione sono "equivalenti": per ogni funzione iterativa si può costruire una equivalente funzione ricorsiva. Per ogni funzione ricorsiva si può costruire una equivalente funzione iterativa.

10 Strategie algoritmiche.

Esistono uan serie di approcci standard per la creazione di algoritmi:

- Divide et impera: dividere il problema in sotto problemi più semplici così da poi unire le soluzioni.
- Greedy (avido): risolvere i problemi più immediati senza preoccuparsi della struttura generale.
- Programmazione dinamica.

10.1 Divide et impera.

Si divide un problema in due o più sottoproblemi più piccoli. Si continua a dividere finché il problema non è facilmente risolubile. Si combinano le soluzioni dei sottoproblemi per produrre la soluzione del problema principale. Un tipo di tale algoritmo si è già visto con l'ordinamento di un array.

10.2 Greedy.

Si costruisce la soluzione di un problema un pezzo alla volta. Ad ogni passo o iterazione si sceglie la soluzione localmente migliore. Si ottiene la soluzione ottimale facendo solo scelte localmente ottimali. Non funziona sempre. Un esempio è l'algoritmo di Dijkstra.

11 Grafi.

Un grafo è un insieme di nodi collegati da archi potenzialmente pesati o orientati.
Bisogna essere in grado di salvare i nodi e gli archi. Si assume che i nodi abbiano nomi $\{0, \ldots, n-1\}$. Ci sono due rappresentazioni: le matrici di adiacenza, le liste di adiacenza. Si assume che il grafo sia statico.

11.1 Matrice di adiacenza.

La sequenza di nodi si può rappresentare come matrice dove la riga in cima indica il nodo destinazione e la colonna più a sinistra indica il nodo sorgente:

	0	1	2	3
0	0	0	0	0
1	1	0	1	0
2	0	0	0	1
3	0	0	1	0

L'arco orientato $(2, 3)$ (da 2 a 3) è presente.

Alcune caratteristiche:

- Veloce (tempo costante) stabilire se un arco esiste.
- Occupazione quadratica di memoria rispetto al numero di vertici.
- Funziona bene per grafi densi.

11.2 Lista di adiacenza.

La lista di adiacenza è una lista di puntatori che puntano ad altri nodi. Nel caso in cui non si punti a nessun nodo, il puntatore assume il valore `NULL`. Un vantaggio è quello di risparmiare memoria rispetto alle matrici.

Alcune caratteristiche:

- Lento (serve scandira una lista) stabilire se un arco esiste.
- Occupazione lineare di memoria rispetto al numero di vertici e archi.
- Funziona bene per grafi sparsi.

11.3 Algoritmo di Dijkstra.

Dato un grafo (con pesi positivi) si vuole trovare il percorso più corto tra due vertici. Alcuni esempi di applicazioni sono il percorso più veloce tra due luoghi e il routing di pacchetti di informazioni.
Si tiene una lista di nodi non visitati. Il nodo iniziale ha distanza 0 e tutti gli altri infinito (o comunque maggiore della somma dei pesi). Si prende il nodo v con distanza minima tra quelli non visitati. Per ogni vicino si aggiorna la distanza vedendo se si può raggiungerlo più velocemente passando da v. Si rimuove v dalla lista dei nodi non visitati. Si continua finché non si può più aggiornare le distanze dei nodi. Si parte dal nodo 1. Il peso è zero e lo si estrae dalla lista dei nodi non visitati. Da 1 si può raggiungere 0 oppure 2, con pesi 2 e 4 rispettivamente. Il nodo con peso minore è 0 e lo si estrae dalla lista dei nodi non visitati. Dal nodo 0 si procede allo stesso modo. Considerando il peso minore per arrivare ad ogni nodo.

12 Linguaggi interpretati.

Alcuni linguaggi non utilizzano un compilatore, bensì il codice sorgente viene interpretato. L'interprete è il programma in esecuzione in memoria, gestito dal sistema operativo, che esegue il codice sorgente senza compilarlo. Anche Bash, la shell di comandi solitamente usata, ha un suo linguaggio, e la shell stessa è un intrprete dei comandi. Si può scrivere un programma (script) in bash.

Differenze. Per il compilatore:

- Il compilatore, come gcc, è esso stesso un programma compilato che viene eseguito solo per trasformare il codice sorgente in codice macchina.
- Il sorgente viene analizzato nel suo insieme e convertito per intero in codice macchina.
- Il processo di compilazione per programmi elaborati può richiedere anche molto tempo.
- Una volta compilato (e collegato alle librerie di sistema) il programma può essere eseguito direttamente.
- In genere, i programmi compilati sono i più efficienti e veloci.

- Alcuni esempi sono C e C++.

Per l'interprete:

- L'interprete è esso stesso un programma compilato, che viene lanciato e dunque caricato in memoria per eseguire il codice sorgente.
- Il codice non viene compilato*, ma preso in esame ed eseguito istruzione per istruzione.
- Spesso è a disposizione di una shell per inserire istruzioni che vengono eseguite al volo.
- In genere, i linguaggi interpretati sono più astratti della macchina, più semplici e leggibili da un essere umano, ma anche più lenti a essere eseguiti.
- Alcuni esempi sono: Python, Bash, R, AWK, Julia, PHP, Java*, Javascript.

(*) in effetti, è compilato in bytecode.

Un interprete è un programma che esegue istruzioni scritte in un linguaggio di programmazione (o di scripting), senza necessità di convertire il codice sorgente in linguaggio macchina prima dell'esecuzione. C0p un ampio spettro di possibilità, di meccanismi di funzionamento e di astrazione dalla macchina a seconda degli interpreti. In genere vengono compiuta una o più delle seguenti operazioni: il codice viene intepretato ed eseguito direttamente; oppure il codice viene tradotto in una sua rappresentazione intermedia, e quindi eseguito (Python, Ruby) o caricato in una macchina virtuale (Java).

In senso astratto, un linguaggio di per sé non è né interpretato né compilato, ma è un'astrazione indipendente dalla particolare maniera con cui viene eseguito da una macchina, Tuttavia, si usa comunemente il termine linguaggio compilato o interpretato con rifeirmento alla maniera canonica con cui tale linguaggio viene usato.
Data la loro semplicità, spesso i linguaggi intepretati vengono usati per lo sviluppo rapido, progetti di piccola dimensione o per risolvere problemi in cui lo sviluppo non è affidato a programmatori puri. È appunto il caso di molte applicazioni scientifiche, dove oggigiorno linguaggi interpretati come Python la fanno da padrone.
La gestione della memoria passa per l'interprete, e questo alleggerisce notevolmente le difficoltà del programmatore, soprattuto sulle possibilità di avere tipi dinamici.

12.1 Python.

Esso è un linguaggio di programmazione interpretato, general-purpose ad alto livello. Esso è uno dei linguaggi più diffusi e in continua crescita, attualmente usato in innumerevoli campi. In ambito scientifico è onnipresente, ed è uno strumento che è fondamentale padroneggiare. Esso è stato progettato per enfatizzare la leggibilità del codice, che fa uso dell'indentazione per racchiudere i blocchi di istruzione. I costrutti del linguaggio e la possibilità di un approccio orientato agli oggetti aiutano a scriver ecodici chiari e logici. I tipi di dati nativi del Python sono dinamici. Questo significa che è possibile cambiarne l'impronta in memoria durante l'esecuzione del programma stesso, in altre parole le variabili possono cambiare tipo e dimensione. Supporta il

paradigma di programmazione che si è visto finora, (procedurale, strutturata), ma anche quello orientato agli oggetti (come il C++ ed il Java) e quello funzionale. Ha una vasittà di librerie (moduli) richiamabili per eseguire operazioni in molti contesti diversi. I moduli di Python possono essere scritti e compilati in C, in modo da estenderne le potenzialità con porzioni di operazioni eseguite da librerie compilate più veloci. Ka oruna versuibe rusake akka fube degku abbu 80, primi anni 90, poi passato a Python 2 nel 2000 ed dal 2008 la versione attuale è Python 3.
L'interprete è sempre lo stesso, tuttavia esistono shell alternative a Bash, come iphython3 ed Anaconda.

Negli script di Linux, si mette nella prima riga `#!/usr/bin/python3`, cioè un indizio a quale interprete (in questo caso python3) deve eseguire il codice seguente. Infatti, si può far girare uno script in python utilizzando `python3 nome_script.py`. Nel caso in cui si specifichi già nello script quale interprete usare, si possono modificare le autorizzazioni del programma con `chmod +x nome_script.py` e si può far partire il programma con `./nome_script.py`.

Tipi di dati primitivi. Esistono i numeri, le stringe, le liste, i dizionari, le tuple (tuples) ed i file. Non è necessario implementare oggetti, strutture curando la gestoine della memoria, implementando routine di accesso e ricerca o altre cose che appesantiscono e distolgono l'attezione dal vero fine del programma. I tipi built-in di Python sono sufficienti per la maggior parte degli scopi, e sono implementati in C.
A differenza del C, in cui è necessario dichiarare esplicitamente il tipo di variabile prima di poterla utilizzare, in python il tipo è associato alla variabile al momento dell'assegnamento in automatico. La sinstassi con cui viene espresso il valore determina il tipo.

- Tipo intero `x = 10`.
- Tipo floating point double `x = 10.0`.
- Tipo complesso `x = 10 + 0j`.
- Stringa di caratteri `x = "10"`.
- Lista di un elemento intero `x = [10]`.

Affinità con C. Si trovano gli operatori noti in C per le normali operazioni, così come gli operatori di confronto, identici al C. Gli opeartori logici (booleani) sono le parole in inglese **and**, **or**, **not**. Ci sono operatori in più, per esempio l'elevamento a potenza con due asterischi `2**3 = 8`. Ci sono alcuni opeartori e concetti nuovi, per esempio l'operatore di appartenenza (membership) **in**, che restituisce vero o falso a seconda che un elemento appartenga ad un certo tipo di dato.
Alcune funzioni utile per i numeri sono l'arrotondamento `round(variabile, numero_di_cifre_di_arrotondamento)`, l'estrazione della parte intera `int(x)`, oppure la conversione in stringa `str(x)`.

Liste. Nelle liste si possono selezionare intervalli di elementi `lista[indice_partenza:indice_arrivo]`, l'indice di partenza è incluso, l'indice di arrivo è escluso.
Ciò che vale per le liste vale anche per le stringe, per le quali si hanno anche altri metodi come `capitalize()` che mette la prima lettera maiuscola, `swapcase()` che inverte maiuscolo con minuscolo.
Per le liste è utilizzato lo slicing `lista[partenza:arrivo:step]`, indici negativi indicato la partenza dalla fine della lista: `lista[-5:]` dal qintultimo elemento fino alla fine.
Altri metodi utili delle liste sono `sort()`, `reverse()`, `append()` nota come push nello stack, `pop()`, `instert(indice, valore)`, `remove(valore_da_rimuovere)`, `count(valore)`, `index(valore)`.
Si possono costruire liste di liste.
Le funzioni `filter(f, x)` e `map(f, x)` hanno come argumenti una funzione (matematica) `f` ed una lista `x`. Per `filter` la funzione matematica deve resituire vero o falso, per `map` una funzione matematica comunemente conosciuta.

Moduli. Per i nomi delle funzioni si usa la regola LGB: si cerca localmente, poi globalmente e poi built-in di python.
In python gli argometni sono passati by object assignment, ovvero assegnatndo gli oggetti ani moni locali. Ciò che viene passato alla funzione sono delle epsressioni i cui valori sono degli oggetti. Gli assegnamenti effettuati dentro la funzione non influenzano il codice chiamante. Cambiando un oggetto modificabile dentro una funzione si può influenzare il codice chiamante (gli argomenti immutabili si comportano come al C "by value", quelli modificabili al C "by reference"). Eventualmente, si può passare una copia dell'oggetto.

Classi. Si possono definire delle classi. I valori assegnati sono gli attributi, le funzioni sono i metodi.

Moduli scientifici. I moduli permettono di estendere le funzionalità di python. Servono a riutilizzare del codice. Essi definiscono lo spazio dei nomi: bisogna scrivere il nome del modulo, punto, poi il nome della funzione nel modulo.
Alcuni moduli scientifici sono `numpy`, `matplotlib`, `scipy`, `pandas` e molti altri.

Esempio. Numpy. Matplotlib. Caricare in python dei dati.

13 Note finali.

13.1 Librerie, hearder file, più file in C.

Si supponga che un programma sia composto da più file. Il file principale `mio_programma.c` è: Mentre `stampa.c` è: E `stampa.h` è: Si compilano tutti i file insieme con `gcc stampa.c mio_programma.c -o mio_programma.exe`.