Fondamenti di Informatica

Banfi Tommaso Felice

September 2022 - January 2023

Contents

Abstract

Sono uno studente di Ingegneria informatica al Politecnico di Milano. Questi sono gli appunti di Fondamenti di Informatica, ovvero Fondamenti di Informatica è un insegnamento introduttivo che descrive i concetti base dell'informatica in modo semplice ed organico. Le lezioni si concentrano sulle conoscenze fondanti per lo sviluppo del software, affrontando argomenti teorici e aspetti pratici, questi ultimi legati allo sviluppo di programmi C/C++. L'esame Fondamenti di Informatica del primo semestre (primo anno) è tenuto dalla Professoressa Bolchini Cristiana, viene erogato in italiano e vale 10 CFU.

1 Introduzione al Corso Fondamendi di Informatica

1.1 Obiettivi dell'insegnamento

Corso di introduzione agli aspetti fondamentali dell'informatica, ossia alla risoluzione dei problemi e alla programmazione in linguaggio C, con riferimento all'architettura di massima dei sistemi di calcolo. L'obiettivo del corso è fare in modo che lo studente/la studentessa sia in grado di risolvere problemi mediante lo sviluppo di programmi, avendo acquisito le conoscenze di base (tecniche di astrazione, tipi di dati fondamentali e strutture di controllo, sottoprogrammi, strutture dati dinamiche, cenni di programmazione modulare e cenni di ricorsione). L'attenzione è posta in egual misura agli aspetti concettuali e a quelli sperimentali. L'insegnamento non adotta modalità di didattica innovativa.

1.2 Risultati di apprendimento attesi

- DdD1. Conoscenza e comprensione
  1. Comprensione dei meccanismi di codifica dell'informazione all'interno di un calcolatore
  2. Conoscenza degli elementi basilari dell'architettura di un calcolatore
  3. Comprensione degli elementi di sintassi e semantica del linguaggio di programmazione C
  4. Conoscenza dei principali costrutti usati nei linguaggi di programmazione imperativi per la rappresentazione e manipolazione dei dati
- DdD2. Capacità di applicare conoscenze e comprensione
  1. Progettazione di strutture dati nella memoria del calcolatore allo scopo di rappresentare efficacemente i dati di uno specifico problema
  2. Progettazione di semplici algoritmi a partire dalla specifica di un problema Realizzazione dell'algoritmo mediante un programma in linguaggio C
  3. Selezione di schemi di computazione noti e loro applicazione per la soluzione di nuovi problemi
- DdD3. Autonomia di giudizio
  1. Valutazione della correttezza degli algoritmi progettati
  2. Individuazione di strutture dati e algoritmi più adatti a specifici problemi
  3. Confronto benefici e svantaggi di diversi approcci algoritmici alla soluzione di un problema dato
- DdD5. Capacità di apprendimento
  1. Essere in grado di apprendere nuovi linguaggi e paradigmi di programmazione
  2. Essere in grado di utilizzare le conoscenze e competenze apprese per affrontare in maniera metodologica problemi complessi

1.3 Argomenti trattati

1. Concetti introduttivi: Algoritmi, programmi e linguaggi Struttura di massima di un calcolatore e di un sistema informatico (hardware, software, Sistema Operativo) - Catena di programmazione.
2. Logica e codifica binaria dell'informazione (logica proposizionale, operatori logici AND, OR, NOT, leggi di De Morgan), rappresentazione dei numeri interi (base 2, 16, notazione in complemento alla base 2), aritmetica binaria, rappresentazione dei numeri reali (notazione in virgola fissa e in virgola mobile), codifica dei caratteri.
3. Aspetti fondamentali della programmazione (con riferimento al linguaggio C): il linguaggio di programmazione e le esigenze di astrazione, la sintassi, struttura di un programma monomodulo, astrazione sui dati (concetto di tipo e tipi base del linguaggio, operatori e compatibilità, i costruttori di tipo array, struct,puntatori), astrazione sul controllo dell'esecuzione (strutture di controllo condizionali, di selezione, iterative).
4. Sottoprogrammi: sottoprogrammi come astrazione per la realizzazione modulare dei programmi, passaggio dei parametri, dati locali, regole di visibilità, sviluppo top down per raffinamento, ricorsione, supporto a run-time per la gestione della chiamata e ritorno da sottoprogramma (record di attivazione, stack e stack pointer). Strutture dati dinamiche, liste collegate a puntatori. Strutture dati persistenti: i file (concetti, operazioni, organizzazione logica), integrazione tra strutture dati in memoria centrale e su file. Attività di laboratorio

5. L'attività del laboratorio ha lo scopo di rendere familiare allo studente sia l'utilizzo pratico del calcolatore sia i metodi e le tecniche utilizzate nella programmazione dei calcolatori. La frequenza al laboratorio è facoltativa e costituisce un'attività particolarmente importante ai fini dell'apprendimento della materia. L'insegnamento non adotta modalità di didattica innovativa.

1.4 Prerequisiti

Nozioni di base sulle caratteristiche e l'uso del personal computer: interfaccia utente, Internet (uso del browser), posta elettronica.

1.5 Modalità di valutazione

L'esame prevede una verifica scritta di circa due ore su tutti gli argomenti dell'insegnamento. In particolare, l'esame prevede:

- la risoluzione di problemi numerici sulla codifica dell'informazione (Descrittore Dublino DdD1);
- domande di carattere teorico a risposta aperta (Descrittori Dublino DdD1, DdD2,DdD3,DdD5);
- progettazione di semplici strutture dati e algoritmi (Descrittore Dublino DdD2);
- identificazione e valutazione di algoritmi alternativi per risolvere un problema dato (Descrittore Dublino DdD3);
- implementazione di programmi in linguaggio C (Descrittori Dublino DdD1, DdD2, DdD3, DdD5).

L'attività di laboratorio non contribuisce alla valutazione dell'esame.

2 Calcolatore e Software

2.1 Informazioni per programmare

2.1.1 Uso del Calcolatore

- Compilare ed eseguire i programmi da terminale per imparare e vedere gli errori che si commettono nello scrivere codice.
- provare e testare le prorpie soluzioni a casa in modo da arrivare alla leziuone successiva avendo capito gli algoritmi risolutivi.
- Importante saper googlare e filatrare le informazioni trovare (affidarsi a siti noti comer StackOverflow).

2.1.2 Software ausiliari

- Editor di testo: è consigliato l'uso di IDE che non siano a completamento automatico in modo da imparare a scirvere codice, uno valido protrebbe essere Notepad++, Atom o VSCode.
- Compilatore: nel coso utilizzeremo GCC ANSI (specifica C*89).
- Ambiente: l'ambiente di sviluppo nel caso di Windows User è Ubuntu.

2.2 Architettura dell'elaboratore

2.2.1 Modello di Von Neumann

L'architettura di Von Neumann è una tipologia di architettura hardware per computer digitali programmabili a programma memorizzato la quale condivide i dati del programma e le istruzioni del programma nello stesso spazio di memoria, contrapponendosi all'architettura Harvard nella quale invece i dati del programma e le istruzioni del programma sono memorizzati in spazi di memoria distinti.

È necessario un mezzo di comunicazione fra la CPU e la memoria di lavoro (RAM), il modello di Von Newmann fornisce solio un modello di riferimento ad alto livello.
La comunicazione può essere di diversi tipi, ad esempio punto a punto o a Bus. La tipologia di comunicazione scelta per essere applicata nei calcolatori è quella a Bus perchè è flessibile (scalabile facilmente), le periferiche guaste non creano problemi (solo il bus è fondamentale) e non è eccessivanmente costosa. Come prestazioni il fatto che il canale non può supportare più periferiche per volta non è un grosso prblema perchè tanto la CPU ne svolge una per volta.

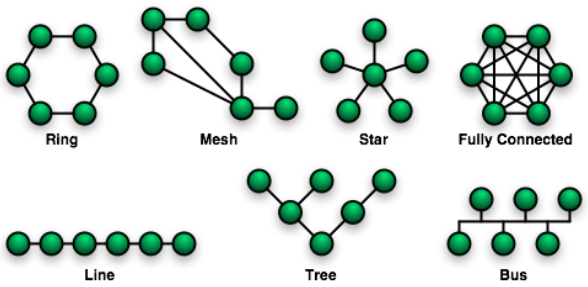

Figure 1: Tipologie della comunicazione

2.2.2 Bus di sistema

Il Bus di sistema è composto da tre livelli:

1. Bus di indirizzi: transitano gli indirizzi di memoria dell'informazione che viene scambiata.
2. Bus di dati: transita l'informazione, quindi o vienee letta la memoria o viene sovrascritta la memoria.

3. Bus di controllo: segnale di controllo (ACK) che avvisa quando è disponibile l'informazione sul bis e coordina le attività dando il via libera alla successiva operazione.

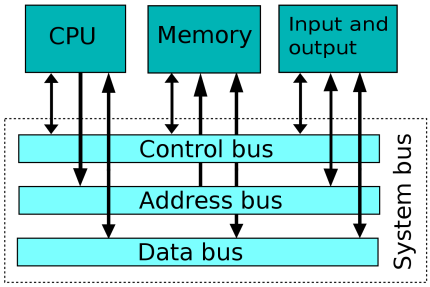

Figure 2: Bus di sistema

La CPU sceglie gli indirizzi che andranno ad interagire sul Bus indirizzi, il dispositivo fa riferimento all'indirizzo sul Bus impostato dalla CPU, infatti:

- il bus indirizzi è monodirezionale dalla CPU alla memoria
- il bus dati è bidirezionale, potrennero essereci perfieriche che possono sia leggere che scrivere in memoria (stampante-scanner), altre che svolgono solo output (schermo monitor) e alcune solo input (tastiera e mouse).

2.2.3 Memoria di lavoro RAM

Nomenclatura:

- Indirizzo di memoria: cella specifica nella memoria
- Parola di memoria: contenuto di una cella di memoria
- Dimensio della parola: dimensione della cella di memoria in numero di bit, gli attuali calcolatori hanno almeno 32 bit (in gneerale per l'aritmentica binaria è comodo un multiplo di 8).

Si può interagire con la RAM in due modi:

- Modalità lettura: si copia la parola ad un certo indirizzo e si mette su bus (possono essere necessari acnhe più giri del bus)
- Modalità scrittura: si copia dal bus e si incolla in un indirizzo.

Se indicheimo con k il numero di bit degli indirizzi, il numero di celle descrivibili sono esattamente 2^k, e qusto viene detto spazio di indirizzamento.

La memoria di lavoro vinee anche chiamata RAM (Random Acess Memory) perchè il tempo di accesso a qualsiasi cella è costante e indipendente dalla posizione della cella in memoria (memoria tabellare), quindi si ha un tempo di accesso sequanziale.

La caratteristica principale della RAM è che è una memoria volatile (non permante), ovvero una volta che non è più alimentata perde il contenuto e all'accensione non si ha un contenuto deterministico (0,1 casuali).
La CPU interagisce direttamente con la RAM.

2.2.4 CPU

La CPU, o unità centrale di elaborazione o processore centrale (central processing unit), nell'architettura di von Neumann di un calcolatore indica un'unità fondamentale (o sottosistema) logico e fisico che sovraintende alle funzionalità logiche di elaborazione principali di un computer.

Il Bus è l'insieme di linee a tensione alta o bassa che corripondono all'1 e 0 che collegano i moduli di un sistema di elaborazione.
La CPU ha il ritmo scandito dal clock che funge da metronomo, la velocità di clock misura il numero di cicli eseguiti dalla CPU ogni secondo (misurati in GHz gigahertz).

Un registro, in informatica e nell'architettura dei calcolatori, è una piccola parte di memoria utilizzata per velocizzare l'esecuzione dei programmi fornendo un accesso rapido ai valori usati più frequentemente:

- **ALU** (arithmetic-logic unit) [U1]:
  Una unità aritmetica e logica (arithmetic and logic unit) è una tipologia particolare di processore digitale che si contraddistingue per essere preposta all'esecuzione di operazioni aritmetiche o logiche.
  L'ALU recupera i dati dai registri del processore (2 ingressi e il selettore), processa i dati nell'accumulatore e provvede a salvare il risultato nel registro di uscita (1 risultato).

- **CU** (control unit) [U2]:
  L'unità di controllo è un componente delle CPU che ha il compito di coordinare tutte le azioni necessarie per l'esecuzione di una istruzione e di insiemi di istruzioni. Le azioni che coordinano i vari settori della CPU.

- **PC** (program counter) [R1]:
  Il program counter è un registro della CPU la cui funzione è quella di conservare l'indirizzo di memoria della prossima istruzione (in linguaggio macchina) da eseguire. È un registro puntatore cioè punta a un dato che si trova in memoria all'indirizzo corrispondente al valore contenuto nel registro stesso.

- **IR** (instruction register) [R2]:
  Il registro istruzione (instruction register) è un registro della CPU che immagazzina l'istruzione in fase di elaborazione. Ogni istruzione viene caricata dentro il registro istruzione che la deposita mentre viene decodificata, la prepara per l'esecuzione e quindi la elabora.

- **SP** (stack pointer) [R3]:
  Lo stack pointer (ESP) è un registro dedicato alla CPU che contiene l'indirizzo della locazione di memoria occupata dal top dello stack. Lo stack viene allocato e deallocato continuamente quindi può esser facile perder traccia della sua "testa".

- **RF** (register file) [R4]: Il register file o banco dei registri è la memoria più vicina al centro di elaborazione in quanto essa sta proprio dentro la CPU.

- **PSW** (process status word) [R5]:
  Il registro di stato (status register o flag register) è un insieme di flag presenti nella CPU che indicano lo stato di diversi risultati di operazioni matematiche. Ci sono diversi flag, un esempio è OF overflow flag.

- **MAR** (memory address register) [R6]:
  Il memory address register (MAR) è un registro temporaneo (buffer) della CPU collegato al Bus indirizzi contenente l'indirizzo della locazione di memoria RAM in cui si andrà a leggere o scrivere un dato. In altre parole, il MAR contiene l'indirizzo di memoria del dato a cui la CPU dovrà accedere. Tutti i risultati di un'elaborazione che devono essere immagazzinati in memoria transitano prima per il registro MDR e da esso raggiungono poi la esatta locazione di memoria.

- **MDR** (memory data register) [R7]:
  Il memory data register (MDR) è un registro a cui la unità aritmetica e logica (ALU) ha accesso diretto e che contiene momentaneamente i dati da/per la CPU. L'MDR, insieme al memory address register (MAR), interfaccia quindi la CPU con la memoria centrale (MC), tutti i dati e le istruzioni che dalla memoria devono essere elaborati nel processore, transitano per il registro MDR e successivamente, da questo, raggiungono gli opportuni registri per l'elaborazione vera e propria.

2.2.5 Memoria di massa

Serve per memorizzare in memoria dei dati che devono esere conservati, infatti essa è una memoria non volatile, quindi se non alimentata non perde il contenuto.
Svolge la funzione di stoccaggio (maagzzino) per tutte le informazioni che non servono nell'immediato.
Come pro ha che è poco costosa e come dimesioni è diversi ordini di grandezza maggiore della RAM, come contro ha che i tempi di accesso sono elevati.

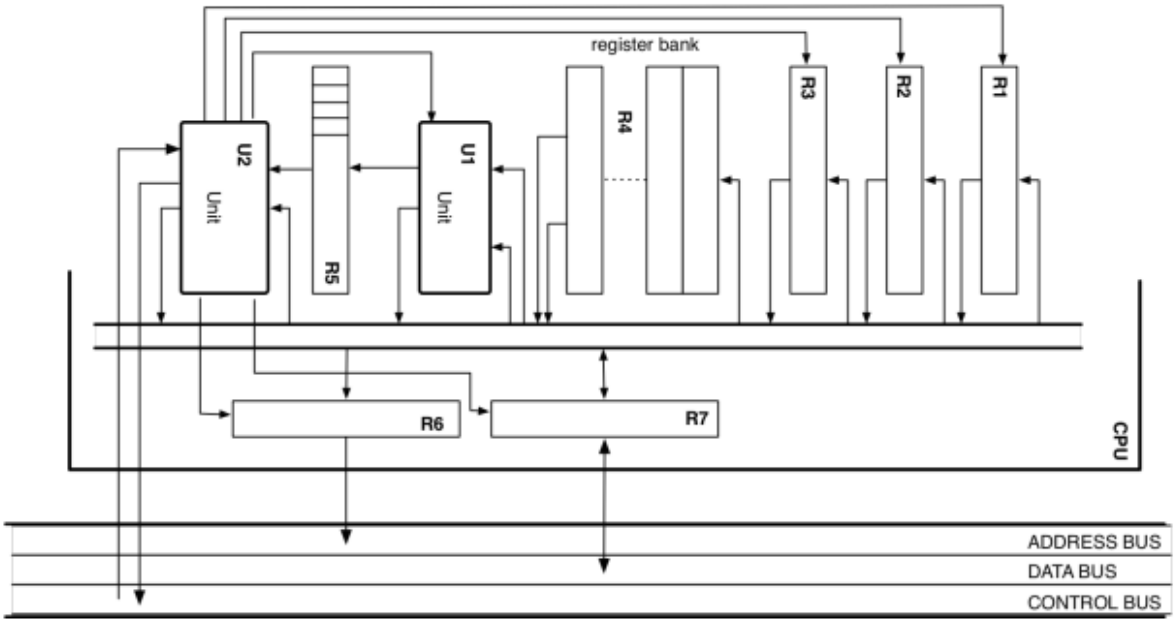

Figure 3: Componenti della CPU

Prima venivano usati gli Hard Disk, ora si usano i dischi a stato solido (SSD) che sono un compromesso fra efficienza e prezzo.

2.2.6 Insieme di istruzioni della CPU

Le istruzioni supportate dalla CPU sono quella basilari, che combinate fra di loro sono in grado di formare istruzioni complesse. Le principali sono:

- Aritemtiche
- Logiche
- Sul flusso di controllo (es: cicli) che infatti iniziano con 'j' che sta per 'Jump'

Nel caso del 'Jump' l'incremento del contatore dell'operazione corrente fa un operazione inutile, ma tutto sommato è più efficiente così perchè si cerca di ottimizzare il caso medio (che spesso è quello che accade più di frequnte) ovvero l'andamento lineare del programma (senza cicli o salti). nel caso di salto si sovrascrive l'indirizzo (gestito dal controllo sul PSW).

Il linguaggio ASSEMBLY è l'unico l'inguaggio interpretato dal calcolatore ed è un insieme di istruzioni composte da due parti:

- Codice operazioni (OP:CODE)
- Operando/i

2.2.7 Gerarchia di memoria

Le diverse memorie di un calcolatore dalla più veloce alla più lenta:

1. Registro:
   Un registro (in inglese: processor register), in informatica e nell'architettura dei calcolatori, è una piccola parte di memoria utilizzata per velocizzare l'esecuzione dei programmi fornendo un accesso rapido ai valori usati più frequentemente e/o tipicamente, i valori correntemente in uso in una determinata parte di un calcolo.

2. Cache:
   La memoria cache (in inglese cache memory, memory cache o CPU cache) è una memoria veloce (rispetto alla memoria principale), relativamente piccola, non visibile al software e completamente gestita dall'hardware, che memorizza i dati più recentemente usati della memoria principale (MM - Main Memory) o memoria di lavoro del sistema.

   La funzione della memoria cache è di velocizzare gli accessi alla memoria principale aumentando le prestazioni del sistema. Nei sistemi multiprocessori con memoria condivisa permette inoltre di ridurre il traffico del bus di sistema e della memoria principale, che rappresenta uno dei maggiori colli di bottiglia di questi sistemi. La memoria cache fa uso della tecnologia veloce SRAM, contro una più lenta DRAM della memoria principale, connessa direttamente al processore.

   Ha un tempo di accesso che aumenta in base alla grandezza della memoria, quindi si deve trovare un balace tra grandezza e prestazioni.

3. RAM: spiegata nel capitoletto prima
4. Memoria di massa: spiegata nel capitoletto prima

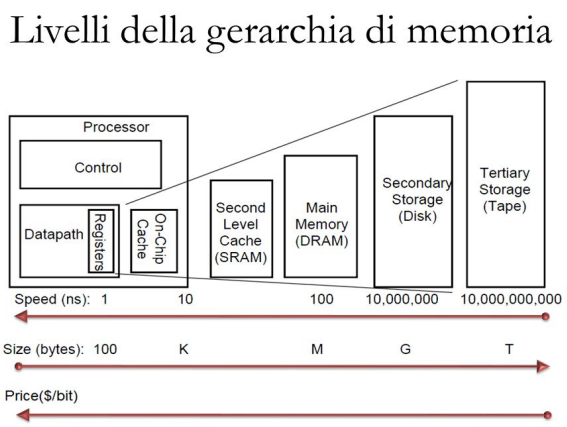

Figure 4: Piramide della gerarchia della memoria

2.3 Organizzazione del sistema operativo

È presente un'organizzazione a livelli:

1. Applicazione (tipo programma in C)
2. Applicazioni che lavorano a basso livello (compilatori, interpreti di comandi, software di sistema)
3. Sistema operativo:
   (a) Gestore dei file
   (b) Gestore delle periferiche
   (c) Gestore della memora lavoro RAM
   (d) Kernel
4. Hardware

La visione dell'organizzazione del sistema operativo viene definita **semplificata ed estesa**:

- Semplificata:
  maschera la complesità dei dettagli delle operazioni richiedendo solo delle istruzioni generali che poi saranno applicate a basso livello con comandi specifici non visibili dall'esterno (nasconde ai programmatori i dettagli dell'hardware fornendogli una API conveniente e facile da usare). Ad esempio per stampare non vengono richieste tutte le specifiche della stampante, basta dire l'operazione richiesta e poi la gestione delle singole operazioni di basso livello vengono svolte

- Estesa:
  il calcolatore non si preoccupa di vedere e valutare la disponibilità delle risorse, sarà poi il kernel che dovrà gestire la CPU per far si che vengano eseguiti tutti i programmi.

2.3.1 Kernel

Separazione tra **meccanismi e politiche**:
Con l'espressione separazione tra meccanismi e politiche nell'ambito dei sistemi operativi si intende la distinzione fra come eseguire qualcosa (meccanismo) e che cosa si debba fare, ovvero quali scelte operare, in risposta ad un certo evento (politica).
Ad esempio nel caso del Kernel il meccanismo è la divisione del tempo della CPU (come gestire la risorsa), mentre la politica è la strategia su come ottimizzare il processo (round robbin).

Spiegazione del Kernel:
C'è uno strato del sistema operativo (più vicino alla macchina fisica) chiamato Kernel che gestisce la CPU. Esso organizza ed esegue i programmi offrendo una visione migliore di quello che la macchina effettivamento ha disponibile al momento, come se rendesse disponibile un processore per ogni processo.
Definizione di processo:

Definition 2.1 (Processo)**.** *un programma quando è in esecuzione viene definito processo, ovvro se il suo stato è attivo e ha dei dati su cui svolgere delle operazioni.*

Chiaramente un calcolatore nessita che vengano eseguiti più programmi in parallelo (o almeno così deve sembrare), quindi viene adottata la politica del **Round Robbin**:

- Divisione del tempo di elaborazione:
  solo una applicazione alla volta può essere attiva, quindi ogni processo ha a disposizione un pò di tempo di esecuzione sulla CPU.
- Stato ready:
  Quando un processo è attivo ed è pronto per essere eseguito è nello stato Ready, ma è in attesa di andare in esecuzione.
- Stato run:
  Il prcesso è in esecuzione sulla CPU perchè èil suo turno e resta in esecuzione per un quantitativo di tempo limitato.
- Stato wait:
  Se al processo serve un dato derivante da un evento esterno non ancora accaduto passa allo stato wait (di attesa) e una volta fornitogli il dato torna in coda come ready.
- Quando il prcesso finisce, o finisce il tempo a suo disposizione sulla CPU (finisce il suo quanto di tempo) passa in lista di attesa come ready to be executed.

La politica di Round Robbin viene definita ad assegnamento ciclico di quanti di tempo, chiaramente non è l'unica politica, ne esistono altre che gestiscono l'assegnamento diversamente (es: priorità ai processi brevi).
Grazie al Round Robbin l'utente avrà l'impressione che più processi siano in esecuzione in parallelo, anche se in realtà il Kernel fa alternare i processi facendoli eseguire parzialmente e un pò ciascuno, ma sempre unon solo per volta in esecuzione (stato run).

2.3.2 Gestione della memoria di lavoro (MMU)

La Memory Managment Unit (MMU), o unità di gestione della memoria, indica una classe di componenti hardware che gestisce le richieste di accesso alla memoria generate dalla CPU.
Aiuta a rendere disponibile ad un processo dei dati nella memoria che hanno nome, posizione e effettiva cella di memoria del dato disaccoppiati.
Infatti si sistemano gli indirizzi di memoria rilocando (attualizzando) gli indirizzi (ad esempio quando finiscono dei processi e si libera dello spazio).

la memoria può essere suddivisa in diversi modi:

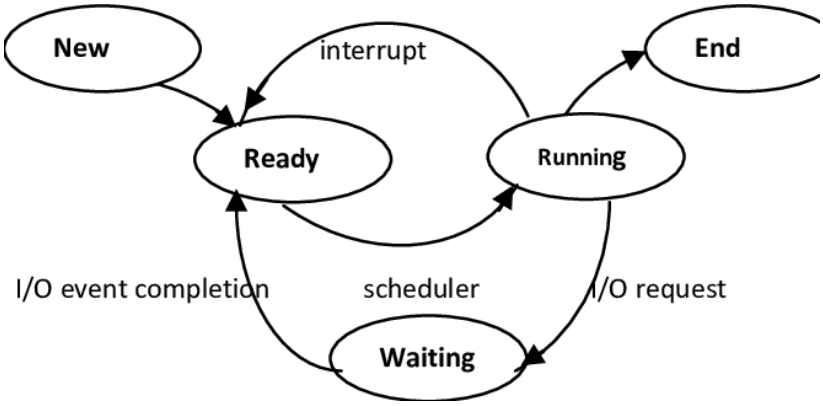

Figure 5: Diagramma dei processi

- Memoria suddivisa a pagine:
  La memoria è suddivisa in pagine di egual dimensione, quindi si ha solo una frammentazione della memoria interna alla pagina.
- Segmentazione della memora:
  La memoria dedicata ai processi viene assegnata in base alla loro dimensione uno dopo l'altro, quindi una volta finiti dei processi si creano dei buchi che vengono riempiti da nuovi processi. Si ha una frammentazione della memoria esterna.

Entrambi i metodi hanno dei pro e dei contro e vengono utilizzati entrambi.

Swapping della memoria:

Definition 2.2 (Swapping della memoria)**.** *passare dal memoria di massa (memoria virtuale) a RAM e viceversa.*

Quando la RAM si riempe perchè ci sono troppi processi pending (programmi in stato ready o wait), ciò che non ci sta nella RAM viene salvato nella memoria virtuale che è una parte di memroia ricavata dalla memoria di massa.
Ora con le SSD lo swapping è molto rapido, prima con gli Hard Disk si notava il rallentamento casuato dalla ricerca nella memoria di massa.

2.3.3 Gestotre delle periferiche

I driver hanno informazioni su come gestire il dispositivo della perifcerica (dipendono dal dispositivo fisico e delle sue singole specifiche.
I driver svolgono principalemnte questi due compiti:

- Gestire microoperazioni a basso livello
- Fornire primitive di accesso semplificate (API)

Ci sono anche meccanismi per periferiche ch creano una cosa e gestiscono le politiche di accesso (sempre con una visione semplificata ed estesa).

Ogni periferica interagisce con sistemi di calcolo con politiche differenti in base al tipo di dato fornito / richiesto:

- Polling:
  Controllo periodico della periferica con una frequanza significativa, quindi serve un buffer per memorizzare informazioni non ancora raccolta
  (es: ogni tot la prof chiede se ci sono domande).
- Interrupt:
  Flag per sollevare un 'interrupt' e segnalare una richiesta di interazione interrompendo il flusso dell'evento che è avvenuto.Viene salvato lo stato e poi mandata la risposta ad interrupt, infine si ripristina lo stato e si prosegue con l'operazione di prima.

(es: se hai domanda chiedi alla prof, ma solo se sono poche se no non riesce a spiegare e fare un discorso).

Per gestire le richieste delle periferiche di accesso alla memoria c'è il **DMA (Direct Memory Access)** che è un coprocessore programmato dalla CPU e viene usato per spostare dati di grosse dimensioni (sposta dalla cella x alla cella y, un numero n di celle).
Il DMA rende il processo molto efficiente perchè non rallenta la CPU, esso lavora solamente nei ritagli di tempo liberi del Bus e così la CPU può fare altro e vengono anche riempiti i tempi morti del Bus.

2.3.4 File System

Il File System è il gestore della memoria di massa e mostra una visione semplificata e logica dei file salvati in memoria permanete (potrebbe anche essere una memoria condivisa).

L'organizzazione è logica ad albero, dove ogni nodo è:

- File e i sui attributi.
- Directory (folder) che è un elemento logico che contiene o file o altre folder.

Di conseguenza dobbiamo avere tutto il percorso (path) per riuscire ad aprire un file salavato nel File System.

Attributi dei file:
Gli attributi di file sono proprietà specifiche di un file, ad esempio:

- A Attributo di file di archivio
- R Attributo di file di sola lettura
- S Attributo di file di sistema
- H Attributo di file nascosto
- I Attributo di file non indicizzato

3 Rappresentazione dell'Informazione

3.1 Scopo della rappresentazione

trovare un modo opportuno per rappresentare all'interno di un sistema di calcolo le informazioni (dati e programmi) in modo efficiente rispetto a:

- Realtà fisica del sistema
- Loro manipolazione

L'efficacia della rappresentazione non bisogna valutarla in base alla facilità di conversione dell'informazione dal modo in cui è rappresentanta al Plain Text comprensibile dall'uomo poichè è irrilevante il tempo di conversione, l'aspetto importante è la facilità di elaborazione.

3.2 Linguaggi

3.2.1 Confronto linguaggio naturale e linguaggio macchina

- Linguaggio naturale (es: italiano): composto da base decimale per numeri e alfabeto per parole.
- Linguaggio macchina:
  - Alfabeto: insieme dei simboli utilizzati ditinguibili tra loro. // S=...
  - Codice: sequenze e simboli di elementi dell'afabeto che hanno un significato particolare e, insieme, compongono l'insieme delle regole del linguaggio (regole definiscono associazione biunivoca tra parole di codice e relative azioni).
  - Cardinalità dell'alfabeto: numero di elementi totali dell'alfabeto, dato dal valore assoluto di S. // —S—
  - Numero di elementi considerati nel messaggio da rappresentare. // n
- Insieme delle configurazioni k:

$$\mathrm{k} = \left\lceil \log_{|S|} n \right\rceil \ ^{1} \tag{1}$$

- Numero di configurazionei diverse di lunghezza k:

$$\mathrm{n} = |S|^{k} \tag{2}$$

3.2.2 Linguaggio del Calcolatore

Nel caso del calcolatore, noi andiamo a considerare il così detto linguaggio macchina, overo quel linguaggio il cui alfabeto è composto da 0 e 1:

- Condensatore scarico/carico.
- linea di tensione alta/bassa.

Si riduce al Codice Binario, ovvero un linguaggio con alfabeto 0, 1 che ha il bit come cifra di codifica (Binary Digit).

Tabella di conversione bit byte kilobyte megabyte

	bit	byte	Kilobyte	Megabyte	Gigabyte
bit	1				
byte	8	1			
Kilobyte	8,192	1,024	1		
Megabyte	8,388,608	1,048,576	1,024	1	
Gigabyte	8,589,934,592	1,073,741,824	1,048,576	1,024	1
Terabyte	8,796,093,022,208	1,099,511,627,776	1,073,741,824	1,048,576	1,024
Petabyte	9,007,199,254,740,990	1,125,899,906,842,620	1,099,511,627,776	1,073,741,824	1,048,576
Exabyte	9,223,372,036,854,780,000	1,152,921,504,606,850,000	1,125,899,906,842,620	1,099,511,627,776	1,073,741,824
Zettabyte	9,444,732,965,739,290,000,000	1,180,591,620,717,410,000,000	1,152,921,504,606,850,000	1,125,899,906,842,620	1,099,511,627,776

Figure 6: Scala conversione Byte

[1] Il simbolo che assomiglia ad una parentesi quadra indica l'arrontondamento per eccesso all'intero più vicino (es: 2.3 =¿ 3)

3.2.3 Esempio linguaggio carte da gioco

- l'insieme degli elementi da rappresentare: i semi delle carte da gioco (4)
- l'insieme delle configurazioni ammissibili
- le regole del gioco delle carte (detto codice) definiscono l'associazione biunivoca =¿ scelta codice binario
  1. S = 0,1
  2. —S— = 2
  3. dimensione delle configurazioni:

$$\left\lceil \log_{|S|} n \right\rceil = \lceil \log_2 4 \rceil = 2 \tag{3}$$

- l'insieme delle configurazioni ammissibil 00, 01, 10, 11 viene associato ai 4 semi delle carte da gioco picche, fiori, cuori, quandri mediante il codice (associazione biunivoca).

3.2.4 Scelta della codifica

Gli aspetti principali da considerare sono i seguenti:

- Insieme degli elementi da rappresentare
- Adozione di una codifica che semplifichi le operazioni che si svolgono più di frequente
- Adozione di una codifica che "conservi" ove utile le proprietà dell'insieme degli elementi da rappresentare (es: elementi adiacenti abbiano codifiche adiacenti)

3.2.5 Tipologie informazioni da rappresentare

Le tipologie di infromazioni che possiamo gestire e analizzare sono le seguebti illustrate nell'immagine. Nello specifico, nel corso di Fondamenti di Informatica andremo a trattare solamente le i dati e infromazioni delle tipologie presenti all'interno dell'insieme rosso.

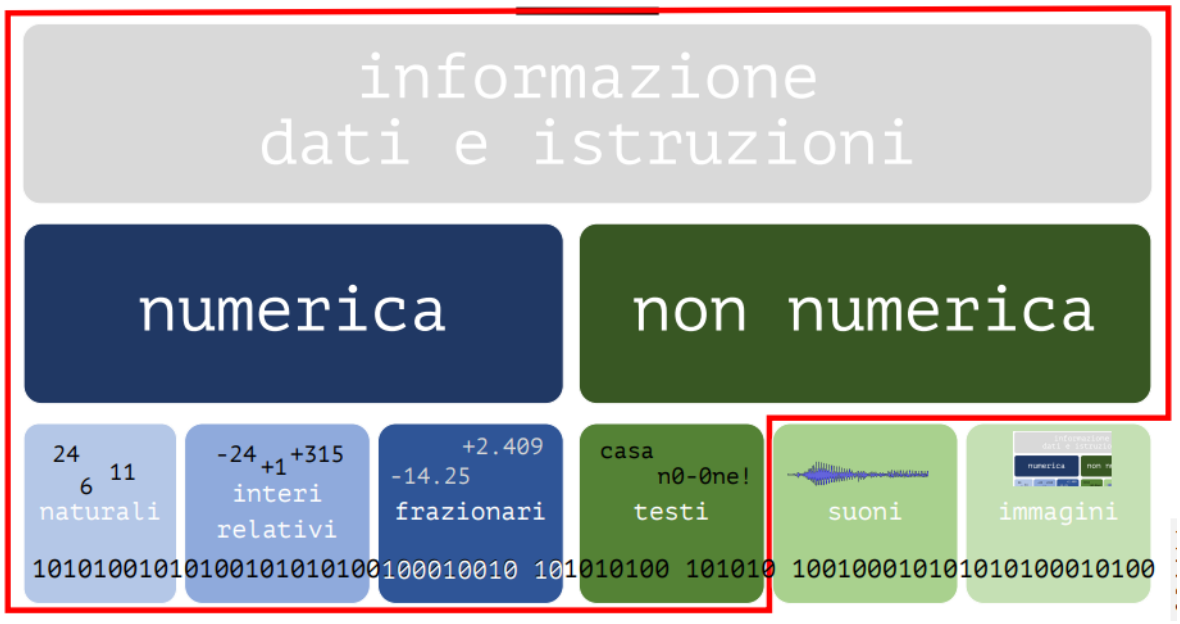

Figure 7: Tipologie di informazioni da rappresentare

- Tipologie trattate: numerico (naturale, intero relativo, frazionario) - non numerico (testi)
- Tipologie non trattate: non numerico (suono, immagine)

3.3 Sistema numerico binario

3.3.1 Definizione

13 Settembre 2022

Definition 3.1. *Il sistema binario, o sistema numerico binario, è un sistema di numerazione posizionale in base 2. A differenza del sistema decimale (in base 10) le uniche cifre che compongono i numeri sono 0 ed 1, e per tale motivo essi vengono detti numeri binari.*

Il sistema binario ha cardinalità 2 poichè l'alfabeto degli elementi dell'insieme è $A = \{0, 1\}$ e ha una notazione posizionale (conta la posizione delle cifre) e pesata (b_i) (a seconda della posizione le cifre hanno un peso diverso).

$$v(N) = \alpha_{n-1}b^{n-1} + \alpha_{n-2}b^{n-2} + \ldots + \alpha_0 b^0 = \sum_{i=0}^{n-1} \alpha_i b^i \tag{4}$$

Nella rappresentazione posizionale, la cifra più a sinistra. Nella rappresentazione posizionale, la cifra più a sinistra prende il nome di cifra più significativa in quanto ha peso massimo, quella più a destra prende il nome di cifra più significativa in quanto ha peso massimo, quella più a destra ($\alpha 0$) di cifra meno significativa, in inglese rispettivamente Most Significant Digit (MSD) e Least Significant Digit (LSD).

3.3.2 Numeri naturali

$$v(N) = \sum_{i=0}^{n} c_i b^i \tag{5}$$

Di seguito riporto la tabella con i numeri decimali e i corrispondenti binari:

Base 10	Base 2
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Possiamo notare che esiste una corrispondenza biunivoca (definita da un codice) fra numero in base 10 e numero in base 2. Per rappresentare un numero n, in realtà si definiscono tutti i valori $0 \leq n \leq numero$ (formula 1).

Gli zeri a sinistra non sono significativi e un numero binario è pari se il bit meno significativo è 0, mentre se è 1 è dispari.

3.3.3 Metodo di conversione base 2

Per convertire da decimale a binario e viceversa sfruttiamo il fatto che esse siano entrambi delle scale posizionali e pesate, ora mostro i due casi:

- Base 10 $\implies$ Base 2: utilizzando la formula 5 possiamo trovare il valore, il procedimento (metodo) ha validità generale e posssono essere applicate a qualsiasi conversione $A \implies B$, ad esempio nel caso $356_2 = (6 \cdot 2^0 + 5 \cdot 2^1 + 3 \cdot 2^2)$
- Base 2 $\implies$ Base 10: il numero decimale viene scomposto dividendo ripetutamente per 2 come nella seguente tabella, il risulate della conversione bisogna leggerlo dal bit più basso a quello più alto ($MSD \implies LSD$):

:2	
17	1
8	0
4	0
2	0
2	0
1	1

:2	
35	1
17	1
8	0
4	0
2	0
2	0
1	1

La conversione del numero 17_{10} in base 2 è 10001_2.

3.3.4 Numeri relativi

Nella notazione MS (modulo e segno) possiamo rappresentare sia i numeri interi positivi che negativi grazie all'introduzione di un ulteriore bit che indica il segno (0 positivo e 1 negativo) del numero rappresentato, ad esempio la conversione del numero -35_{10MS} corrisponde a:

:2	
35	1
17	1
8	0
4	0
2	0
2	0
1	1

La conversione del numero 34_{10} in base 2 è "100011_2", che in notazione MS è "$0\,100011_2$", noi stavamo convertendo -35_{10MS} quindi dobbiamo farlo diventare negativo: "$1\,100011_{2MS}$".

È importante segnare con l'apice la notazione con cui stiamo esprimendo il numero perchè, ad esempio il numero 1011 potrebbe essere interpretato come 1011_{10}, 1011_2 (11_{10}) e come 1011_{MS} (-2_{MS}). Per quanto riguaqrda il calcolatore, esso interpreta tutto in nella notazione 2C2 perchè ha un'aritmetica estremamente efficiente ed ottimizzata che vedremo in seguito.

3.3.5 Notazione complemento alla base

Il complemento a due, o complemento alla base, è il metodo più diffuso per la rappresentazione dei numeri con segno in informatica, questo perchè l'aritmetica è estemamente ottimizzata (si evita il problema dell'analisi del segno dei numeri nelle somme). In questa notazione se si somma $x_{2C2} + (-x_{2C2}) = 0000$. Ad esempio avendo a disposizione 4 bit, in notazione complemento a 2 si ottiene:

Base 10	Base 2	2C2
+0	0000	+0
+1	0001	+1
+2	0010	+2
+3	0011	+3
+4	0100	+4
+5	0101	+5
+6	0110	+6
+7	0111	+7
-0	1000	-0
-1	1001	-7
-2	1010	-6
-3	1011	-5
-4	1100	-4
-5	1101	-3
-6	1110	-2
-7	1111	-1

Si nota che c'è una rindondanza perchè compare sia -0 che +0, che viene sistemata assegnando al valore in 2C2 di -0 il valore di -8 (per quanto riguarda l'aritmetica interna il valore può essere ± 8 perchè è circolare, ma per convenzione del MSD=1 si considera -8).
Analogamente esiste il complemento alla base 3, è importante sottolineare che, essendo la base dipari, non vale più la regola dello $MSD(0) \implies positivo$ e $MSD(1) = pari$; un altro aspetto particolare delle basi dispari è che, secondo la formula 1 delle configurazioni, esse non hanno l'inefficienza del doppio zero (± 0) che nelle basi pari viene gestito due volte, sia come negativo che positivo.

3.3.6 Somma e differenza in complemento a 2

Per essere sommati due numeri devono avere la stessa configurazione di bit, quindi nel caso uno fosse espresso con meno bit bisogna estenderlo senza farlo cambiare di valore, per fare ciò si ripete il bit più significativo (primo da sinistra).

I valori in complemento a 2 sono calcolati trovando il complementare che sommato ad essi dia 0000, così facendo l'aritmetica è ottimale e rapida in modo da rendere veloce l'elaborazione. Di conseguenza in questa notazione non esiste il concetto di segno e modulo, perchè essi sono già inclusi nel numero; possiamo affermare che se il bit più significativo è 1 il numero sarà negativo, viceversa se il MSD è 0 il numero sarà negativo.

3.3.7 Operazioni aritmetiche

L'aspetto più imposrtante da considerare in un sistema numerico su cui basare un calcolatore è l'ottimizzazione dell'aritmetica in tale base. In base binaria le operazioni svolte possono essere ricondotte e poche operazioni base che non necessitano di molte condizioni e ragionamenti. Di seguito ci somo degli esempi di operazioni in base 2:

- Somma (+): i numeri devono essere delle stesse dimensioni e vengono sommati bit a bit tenendo il riporto nel caso un bit risulti essere maggiore di 1:
- Differenza (-): la differenza è di fatto la somma del numero cambiato di segno (opposto del numero), quindi basta cambiare di segno il secondo termine per ritornare nel caso precedente di somma:

$$\begin{array}{ll} 100111 & + \\ 011100 & = \\ \hline 000011 & \end{array}$$

Chiaramente questo metodo diventa estremamente efficiente quando è molto facile ricavare l'opposto di un numero, ovvero quel numero $x \in \mathbf{Z} \,|\, x + (-x) = 0$, per questo motivo i calcolatori operano con la notaione complemento a 2.

3.3.8 Overflow

Definition 3.2. *Con overflow si intende il superamento della capacità massima di memoria o di operatività di un computer, che è causa di errore.*

L'overflow di per sè, all'interno dell'ambiente di un calcolatore, non è un problema e l'artimetica funziona corretamente come si può vedere nell'immagine ref(fig:overflow) perchè viene trascurato il bit in eccesso. Il problema

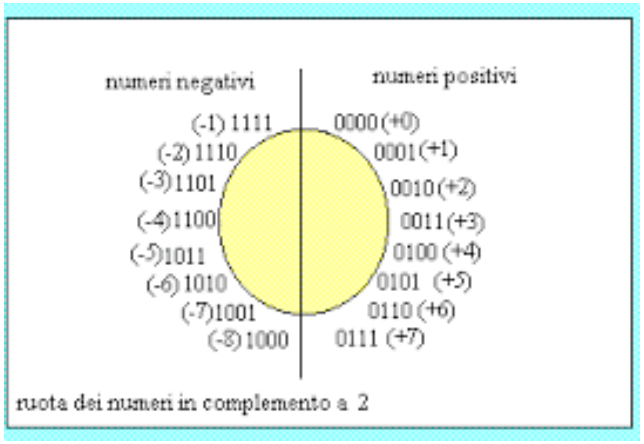

Figure 8: Aritmetica nel calcolatore

nasce quando il calcolatore deve trasmettere delle informazioni all'esterno dove è importante il valore effettivo del dato, quindi il fatto che il bit più significativo vengo troncato influisce sul numero che viene trasmesso nel mondo esterno al calcolatore (nella conversione da 2C2 a base 10). Per stabilire se si è verificato overlow si devono considerare due casistiche, ovvero quando i due elementi sommati sono:

- Concordi: se il risultato è discorde con i due numeri sommati (es: $+2+4=-3$) si è verificato overlow, ovvero quando il risultato non è plausibile .
- Discordi: non si può verificare overflow.

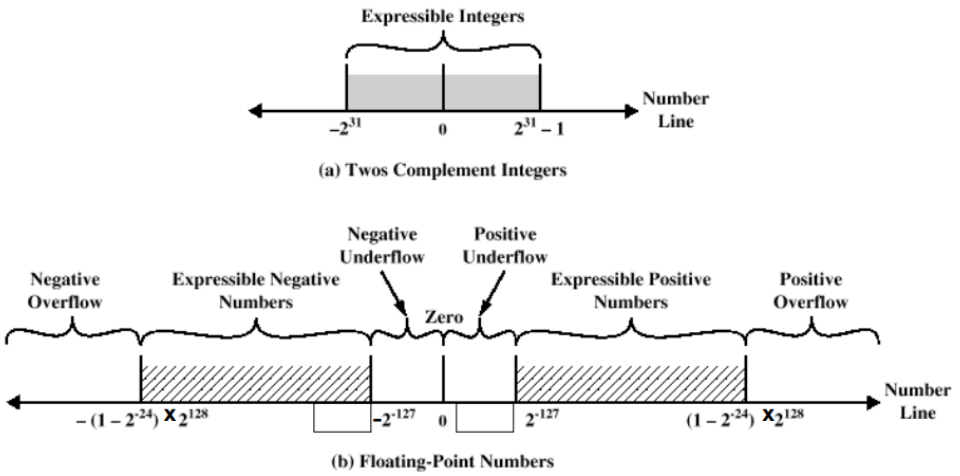

Figure 9: Linea dei numeri ed overflow

3.3.9 Metodo di conversione 2C2

- Notazione $10_{MS} \implies$ 2C2:
  - Positivo ($MSD = 0$):
    1. Applicare il procedimento classico di conversione di un numero in base 2
    2. Scomporre dividendo per 2
    3. Leggere il numero ottenuto nella scomposizione dal basso verso l'alto ($MSD \implies LSD$)
  - Negativo ($MSD = 1$):
    1. Trasfromare il numero negativo x_{10MS} in $-x_{10MS}$, che consiste nel trovare l'opposto (cambio segno)
    2. Applicare il procedimento classico di conversione di un numero in base 2
    3. Scomporre dividendo per 2
    4. Leggere il numero ottenuto nella scomposizione dal basso verso l'alto ($MSD \implies LSD$)
    5. Ricordarsi di cambiere segno nel risultato finale perchè noi abbiamo lavorato con $-x_{10MS}$ e noi dovevamo convertire x_{10MS}
- Notazione 2C2 $\implies 10_{MS}$:
  - Positivo ($MSD = 0$):
    1. Moltiplicare per 2 elevato alla posizione di ogni cifra secondo la formula 5
    2. Sommare i risultati parziali e si ottiene il numero in notazione 10_{MS}
  - Negativo ($MSD = 1$):
    1. Trasfromare il numero negativo x_{2c2} in $-x_{2c2}$, che consiste nel trovare il complementare
    2. Moltiplicare per 2 elevato alla posizione di ogni cifra secondo la formula 5
    3. Sommare i risultati parziali e si ottiene il numero in notazione 10_{MS}
    4. Ricordarsi di cambiere segno nel risultato finale perchè noi abbiamo lavorato con $-x_{2c2}$ e noi dovevamo convertire x_{2c2}

3.3.10 Rappresentazione valori numerici razionali

Considerato un numero razionale $\frac{m}{n} \quad , \quad n \in \mathbf{N} - \{o\}$, vale la stessa regola dei numeri interi in base 2, ad esempio:

$$101.01_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 4 + 0 + 1 + 0 + \frac{1}{4} = 5,25 \tag{6}$$

Nella scrittura del decimale si deve definire quanti bit vanno a rappresentare la parte intera e quanti quella decimale. Esisteono due tipologie di notazioni per rappresentare i numeri razionali:

- Notazione con virgola fissa: in questa notazione vengono stabiliti e fissati il numero di bit detinati alla rappreswnazione della aprte intera e a quella decimale, ad esempio in 101101_2 possiamo decidere arbitrariamente che $\underbrace{10}_{\text{intero}}\ \underbrace{1101}_{\text{decimale}}$, che si sicrive 10.1101.

  Per la formula delle configurazioni 1 sappiamo che se, ad esempio, destiniamo 2 bit abbaimo 4 configurazioni disponibili quindi fra un numero intero n e il suo successivo $n+1$ ci sono $4-1$ numeri rappresentabili.

  Porprio per questo, a differenza degli interi che o il numero è corretto o si è verificato overflow, i numeri razionali possono o essere rappresentati correttamente o il numero viene approssimato (troncato) alla configurazione vicina (precisione vincolata al numero di bit, ovvero alle configurazioni).

  Nella notazione a virgola fissa l'errore assoluto rimane costante, mentre è variabile l'errore relativo come si può vedere nella formula $\varepsilon_R = \frac{\varepsilon_A}{valore}$.

- Notazione con virgola mobile: in questa notazione viene dedicato il numero minimo di bit alla parte intera del numero da rappresentare e il resto dei bit discponibili viene riservata alla parte decimale.

  Così facendo la firgola viene spostata, da qui il nome virgola mobile (floating point), dove è ottimale e si sfruttano tutti i bit disponibili per avere una precisione della parte decimale massima. La virgola mobile la ritroveremo in programmazione quando utilizzeremo le variabili di tipo $float$.

  Nella notazione a virgola mobile l'errore relativo rimane costante, mentre è variabile l'errore assoluto.

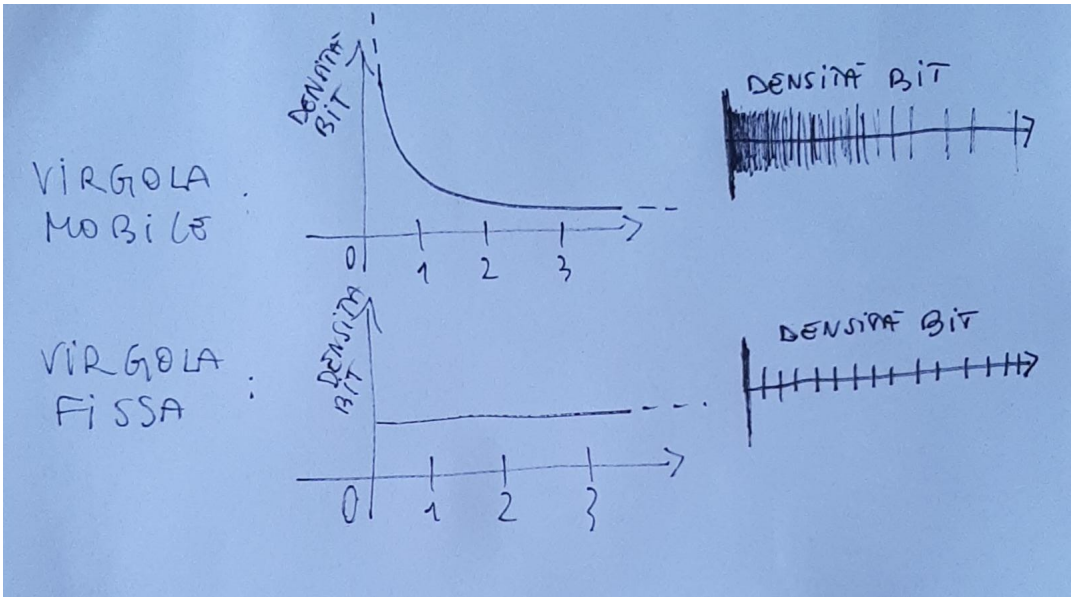

Figure 10: Distribuzione bit in virgola fissa (sopra) e mobile (sotto)

Come per i numeri interi, anche per i numeri razionali valgono le stesse regole di converisone che si basano sulla formula 5, un esempio di esercizio potrebbe essere trasformare il numero $13.75_{\{10\}}$ in base 2 comolemento a 2:

:2	
13	1
6	0
3	1
1	1

x2	
0.75	/
0.75*2=1.50	1
0.50*2=1.00	1
0.00	0

Nella tabella a sinistra si calcola il corrispondente della parte intera dividendo ripetutamente la base (2) e considerando il risultato dal basso verso l'alto ($MSD \implies LSD$), nell'esempio il risultato della parte intera è 1101.

Nella tabella a destra si calcola il corrispondente della parte decimale:

1. si moltiplica per la base, in questo caso 2
2. il risultato ottenuto se è maggiore o uguale a 1 corrisponde a 1, viceversa si scrive zero
3. si considera solo la parte decimale e si ripete il procedimento dal punto 1

4. una volta arrivati a 0.00, il risultato va considerato leggendolo dall'alto verso il basso ($MSD \implies LSD$), nell'esempio il risultato della parte decimale è 110.

Sia nel caso di virgola fissa che mobile si può utilizzare la notazione Modulo Segno (MS), dove 0 identifica un numero positovo e 1 negativo.

3.3.11 Notazione scientifica

I numeri binari possono essere scitti anche in notazione scientifica, questa notazione è utilizzata perchè è utile per riuscire a scrivere dei numeri che richiedono l'impiego di molti bit in maniera più ottimizzata. Ad esempio:

$$13.75_{10} = 1.375 \cdot 10^1 = 1101.110_2 = 1.101110 \cdot 2^3 \tag{7}$$

Il numero in notazione scientifica può essere scritto come:

$$1.M \cdot base^{exp} \tag{8}$$

In questo caso stiamo considerando la base 2, la parte decimale è rapresentata dalla mantissa e infine l'1 è implicito e non necessita di essere salvato sprecando bit. L'esponente viene ritoccato per renderlo sempre positivo, gli viene sommato il numero di configurazioni negative diposnibili meno uno in base ai bit a disposizione in modo da evitare che si presenti un numero negativo ad esponente, ad esempio se ci sono 8 bit, allora la somma sarà di $2^8 - 1 = 255$.

Segno	Esponente	Mantissa

Lo Standard della notazione scientifica nei calcolatori è IEEE754 e definisce come vengono storate le informazioni numeriche binarie in notazione scientifca. Esistono due tipologie di notazioni in base alla precisione necessaria:

- Singola precisione:

+/−	Exp	Mantissa
1	8	23

32 bit totali

- Doppia precisione:

+/−	Exp	Mantissa
1	11	52

64 bit totali

Lo standard IEEE 754 della notazione scientifica attribuisce valori convenzionali a particolari configurazioni di esponente e mantissa:

Normalizzato	±	0<esp<Max	Qualsiasi stringa di bit
Denormalizzato	±	0	Qualsiasi stringa di bit diversa da zero
Zero	±	0	0
Infinito	±	111…1	0
NaN	±	111…1	Qualsiasi stringa di bit diversa da zero

Figure 11: Valori convenzionali notazione scientifica

3.3.12 Sistema numerico esadecimale

Nell'informatica è importante il sistema numerico in base 16 (esadecimale) perchè semplifica estremamente la conversione avendo la base come potenza della base 2 ($16 = 2^4$) le conversioni vengono svolte di 4bit in 4bit e la lettura per l'utente è facilitata (comunicazione tra utenti di informazioni del compilatore).

Ad esempio, per convertire il numero 0101010111011010_2 in base 16 basta:

$$\underbrace{0101}_{5}\underbrace{0010}_{3}\underbrace{1101}_{D}\underbrace{1010}_{A} \tag{9}$$

Ottennedo facilmente e rapidamente il numero convertito in base 16, viceversa si può risalire al numero in base 2 partendo dal numero in base 16.

Decimal	Binary	Hexadecimal
0	0	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Figure 12: Tabella di conversione sitema decimale, binario ed esadecimale

3.4 Rappresentazione informazioni non-numeriche

L'obiettivo è definire una codifica per ogni possibile carattere rappresentabile, quindi è necessario introdurre, oltre alla codifica per i numeri, una codifica per i caratteri testuali.

3.4.1 Alfabeto caratteri

Inizialmente i caratteri da rappresentare erano:

- Alfabeto base a b ... z A B ... Z
- Caratteri numerici 0 1 ... 9
- Simboli interpunzione . , : ...
- Caratteri "speciali" a-capo nuova-riga spazio

In totale questi caratteri sono 120, di conseguenza per la formula delle configurazionei 1 i bit necessari sono:

$$\left\lceil \log_{|S|} n \right\rceil = \lceil \log_2 120 \rceil = 7 \tag{10}$$

3.4.2 Codice ASCII

Per rappresentare l'informazione non numerica uno dei codici più comunemente utilizzati è il codice ASCII (American Standard Code for Information Interchange), basato sul sistema binario e che consta di parole di codice di 7 bit. Il codice ASCII è non ridondante, perchè i simboli codificati sono in numero pari alle configurazioni ottenibili con 7 cifre binarie.

1. 0-32: caratteri non stampabili come simboli di controllo e funzioni varie
2. 48-57: cifre numeriche per base 10
3. 65-90: caratteri alfabeto maiuscoli
4. 97-123: caratteri alfabeto minuscoli

Poichè i caratteri sono però ben più numerosi di 127 anche nel nostro alfabeto, in realtà si utilizza il codice ASCII esteso, che utilizza 8 bit, avendo quindi a disposizione 256 diverse configurazioni. Questa parte della codifica però presenta varie versioni a carattere nazionale, quindi ci sono diverse codifiche per l'intervallo 128-255.

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

Figure 13: Tabella ASCII base

4 Algoritmi

4.1 Calcolatore vs Umano

Il calcolatore è più veloce a svolgere operazioni rispetto ad un umano (quindi più scalabile), è in grado di svolgere calcoli e operazioni estremamente complesse e non commette errori di distrazioni o calcolo. Per far si che il calcoatore possa risolvere dei problemi, è necessario che l'umano capisca come risolvere il problema su "un pezzo di carta", in modo da implementare l'algoritmo risolutovo che il compilatore è in grado di interpretare ed utilizzare.

4.2 Risoluzione di un problema

4.2.1 Modello generale del procedimento

1. Trovare tutti i punti del problema, ovvero fare un elenco di quello che serve.
2. Raccontare a voce/su carta il procedimento risolutivo che reputiamo corretto e ottimale.
3. Trovare modello risolutivo considerando che molto spesso il nostro cervello ragiano già i maniera ottimizzata.

4.2.2 Esempio trovare massimo in lista

1. Scrivere processo risolutivo a parole (algoritmo risolutivo):

```
Devo analizzare NUM = 104 valori
Prendo il primo valore e lo considero come il valore massimo
Ho trattato un valore fino a quando ho analizzato un numero di valori minore di NUM
    Prendo un nuovo valore
    Se è maggiore dell'attuale valore massimo
        Aggiorno il valore massimo a quello appena analizzato
    altrimenti non ho nulla da fare | anche inutile dirlo
    ho trattato un valore in più
dico il valore massimo
```

2. Riscrivere l'algoritmo in un modo simile al codice C (pseudocodice):

```
NUM = 104
acquisisco val
max = val
cont = 1
fino-a-quando cont < NUM
    acquisisco val
    se val > max
        max = val
    cont = cont +1
dico max
```

3. Riscrivere l'algoritmo codice C:

```c
#define NUM 104
int main(int arg, char * argv[])
{
int val, max, cont;
scanf("%d", &val);
max = val;
cont = 1;
while(cont < NUM){
    scanf("%d", &val);
    if(val > max)
        max = val;
```

```
            cont = cont + 1;
        }
        printf("%d\n", max);
        return 0;
        }
```

4.2.3 Ottimizzazione e efficienza

Partendo dalla risoluzione dell'esempio precedente, in modo da evitare ridondanza e perdita di tempo, è utile controllare allo stesso tempo sia MAX che MIN facendo scorrere la lista una sola volta. Controllare MAX e MIN in due momenti separati implicherebbe lo scorrimento doppio della lista, il che è inefficiente e non ottimizzato. L'ottimizzazione è molto impoetante soprattutto per quando il programma va scalato su una mole di dati e operazioni molto grandi. Proprio per questo la programmazione è 80% analisi del problema e il trovare un algoritmo ottimizzato ed efficiente, mentre solo il 20% restante è lo studio della sintassi del singolo linguaggio di programmazione, che nel nostro caso è C. Molto spesso gli algoritmi ammettono solo una soluzione efficace e quindi ottimale
Le regole generali per scrivere degli algoritmi risolutivi ottimizzati sono:

1. Non fa la differenza la grandezza (entro un certo limite) degli operandi perchè è getsita dal calcolatore.
2. Il numero delle operazioni, sia assegnazione che confronto, deve essere limitato allo stretto necessario.
3. Non devono esserci operazioni a vuoto che rallentano il calcolatore e non contribuiscono alla risoluzione.
4. Se il risultato di un controllo che da la soluzione del problema è di tipo booleano (0,1) possiamo stampare direttamente in output il risultato tramite solamente un'assegnazione.

4.3 Algoritmo

Definition 4.1 (Algoritmo)**.** *sequenza di passi (procedimento) che a partire dai dati iniziali in un numero finito di passi non ambigui produce un risultato che costituisce la soluzione del problema.*

Un algoritmo per essere tale deve avere delle caratteristiche ben precise:

- i passi devono essere di numero finito e possono essere suddivisi in blocchi di controllo (es: if) e blocchi operativi (es: assegnazione)
- non deve essere ambiguo
- deve essere deterministico, ovvero a parità di dati di ingresso si deve ottenere lo stesso dato in uscita

Un algoritmo può essere rappresentiato in diversi modi:

- Diagramma di flusso: è una rappresentazione grafica delle operazioni da eseguire per l'esecuzione di un algoritmo.

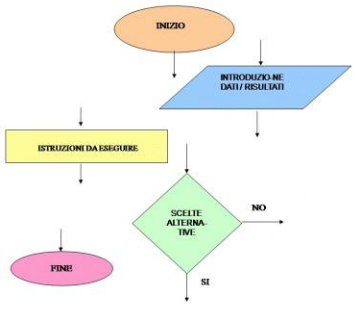

Figure 14: Blocchi dei diagrammi di flusso

- pseudocodice: un linguaggio che ci permette di descrivere programmi usando una sintassi naturale, umana, senza le rigide regole di un linguaggio di programmazione.
- codice: ad esempio potremmo utilizzare C, C++ o Python.

5 Linguaggio C89 ANSI

Lo standard ANSI del linguaggio di programmazione C è stato progettato per promuovere la la portabilità dei programmi C fra una varietà di sistemi informatici. Per realizzare questo, la norma copre tre aree principali: l'ambiente in cui il programma compila ed esegue, la semantica e la sintassi del linguaggio e il contenuto e la semantica di un insieme di routine di libreria e file di intestazione.

5.1 Sintassi linguaggio

5.1.1 Operatori

- Assegnamento "=" che corrisponde ad dire *var* := *qualcosa*, ovvero definisce a cosa corrisponde il valore della variabile con quello che sta a destra dell'uguale, per questo l'assegnamento è unidirezionale, infatti $(A = B) \neq (B = A)$
- Operatori aritmetici (bisogna stare attenti al tipo della variabile perchè in base a quello si svolgono operazioni differenti):
  - Somma "+":
  - Differenza "−":
  - Prodotto "∗":
  - Divisione "/":
  - Resto divisione "%":
- Operatori logici, si seguono le tabelle di verità dei diversi operatori:
  - Prodotto logico (AND) "&&"
  - Somma logica (OR) "||"
  - Negazione (NOT) "!"
- Operatori relazionali:
  - Maggiore stretto ">" // Maggiore o uguale "≥"
  - Minore stretto ">" // Minore o uguale "≤"
  - Confronto "==" per ditinguerlo da assegnamento ("=0")
  - Diverso "! =" (non uguale)

Ripasso condizioni logiche nel while e if (legge di De Morgan):

- (condiz_1 || condiz_2) == !(!condiz_1 && !condiz_2)
- (condiz_1 && condiz_2) == !(!condiz_1 || !condiz_2)

5.2 Compilare ed eseguire

L'estensione del file è fondamentale perchè comunica al compilatore il probabile contenuto del file, ad asempio se salviamo un file come nomefile.c permettiamo all'editor di riconoscere il linguaggio di programmazione C ed evidenziare le keyword.
In qesto corso per gli studenti che utilizzano un compilatore windows sarà necessario installare una virtual machine di linux (ad esempio Ubunto for windows) che permette di eseguire i programmi da riga di comando di linux.

Per compilare un file si scrive nel prompt dei comandi di ubuntu:
gcc -Wall -std=c89 -pedantic -o nomefileeseguibile nomefilesorgente Analizziamo questo comando (l'ordine dei parametri non è importante e si possono scambiare fra loro):

- -Wall segnala tutti i warning
- -std=c89 controlla che il codice rispetti lo standard ANSI c89 del linguaggio C
- -pedantic segnala le violazioni segnalate
- -o nomefileeseguibile è il file eseguibile (in windows ha estensione file.exe e su linux non ha estensione)

- -nomefilesorgente è il file sogente .c

Si possono presetare due problemi:

- Error: non si crea l'eseguibile, abbiamo sbagliato sintassi (indicano anche riga codice) se si dimentica ; errore è nella riga dopo
- Warning: condizioni strane che potrebbero indicate la presenza di un problema, ma viene creato comunque un eseguibile possibile eerore di semantica (non ultima versione gcc)

Per l'esecuzione è necessario specificare al compilatore dove si trova il file eseguibile (directory del file).
Esistono dei tool utilissimi per i neoprogrammatori (sudo apt install nomeprogramma) che sono:

- Indent: formatta e indenta corretamente file.c
- Diff: confronta due file evidenziando i caratteri differenti

5.2.1 Spiegazione sintassi codice

```c
/* (1) inclusione librerie */
#include <stdio.h>

# define TOM 99.99

/* (2) int main */
int main(int argc, char*argv[])
{
    /* (3) indentazione*/
    /* (4) dichiarazione variabili */
    int intero1, intero2 , intero3;
    float decimale, tot;
    char carattere;

    /* (5) input */
    scanf("inserisci numero: %d \t %d", &intero1, &intero2);
    scanf("inserisci numero:\n %f", &decimale);

    /* (6) assegnamento (elaborazione) */
    tot = intero1 + intero2 + decimale;

    /* (7) output */
    printf("il totale è:\t", tot);

    return 0;
}
```

Spiegazione delle parti del programma:

1. Librerie: la libreria standard del C è un insieme di librerie che forniscono al programmatore funzioni tipizzate. Insieme alla libreria vengono inoltre forniti gli Header file, file di testo che permettono al programmatore di utilizzare lo specifico insieme di funzioni della libreria ad esse associate.
2. Int main: la funzione main è il punto di inizio per l'esecuzione di un programma. Essa è generalmente la prima funzione eseguita durante l'avvio di un programma.
3. indentazione: è l'inserimento di una certa quantità di spazio vuoto all'inizio di una riga di testo (tab = 4 spazi), è una convenzione utilizzata per esprimere al meglio la struttura di un codice sorgente. In alcuni linguaggi come C e C++ è una convenzione, mentre in altri come Python fa parte della sintassi.
4. Dichiarazione variabili: è importante per stabilire quanto spazio deve riservare il calcolatore per un certo dato e per sapere la tipologia di dato che deve essere pronto a ricevere.
5. Input: la sintassi è 'scanf("%tipo_variabile", &nome_variabile);', si possono leggere anche più di una variabile alla volta. I tipi della variabile possono essere molteplici e vengono spiegati nella foto 16.

6. Assegnamento ed elaborazione: vongono svolte operazioni di assegnamento, confronto, ecc usando gli operatori consentiti.

7. Output: la sintassi è 'printf("testo in output: %tipo_dato", nome_variabile);' e possiamo usare i caratteri ASCII (13) non stampabili nell'immagine 17.

TIPO	N. BIT	VALORI RAPPRESENTABILI
char	8	Da -128 a +127 (o caratteri ASCII corrispondenti)
unsigned char	8	Da 0 a 255 (o caratteri ASCII corrispondenti)
signed char	8	Gli stessi del tipo base "char"
int	16	Da -32768 a +32767
unsigned int	16	Da 0 a 65535
signed int	16	Gli stessi del tipo base "int"
short int	16	Gli stessi del tipo base "int"
unsigned short int	16	Gli stessi del tipo "unsigned int"
signed short int	16	Gli stessi del tipo base "int"
long int	32	Da -2147483648 a +2147483647
signed long int	32	Gli stessi del tipo "long int"
unsigned long int	32	Da 0 a 4294967295
float	32	Circa da $-3{,}4\times10^{38}$ a $+3{,}4\times10^{38}$
double	64	Circa da $-1{,}8\times10^{308}$ a $+1{,}8\times10^{308}$
long double	80	Circa da $-1{,}2\times10^{4932}$ a $+1{,}2\times10^{4932}$

Figure 15: Tipi delle variabili in C

%d	per il tipo int
%ld	per il tipo long int
%f	per il tipo float
%lf	per il tipo double
%c	per il tipo char
%s	per stringhe di caratteri

Figure 16: Tipi di dato in C

Carattere speciale	Descrizione
\n	Newline o anche ritorno a capo
\r	Carriage return o ritorno a inizio rigo
\t	Tab
\v	Vertical tab
\b	Backspace
\f	Form feed
\a	Alert o beep
\'	Singolo apice
\"	Singole virgolette
\?	Punto interrogativo
\\	Backslash

Figure 17: Caratteri speciali in C

5.3 Funzioni

5.3.1 Sottoprogrammi

Sottoprogrammi sono composti da funzioni e sottoprocedure (termine piu generale).
Ruolo dei sottoprogrammi è spostare parte dell'elaborazione in dei sottoprogrammi scomponendo un problema in tanti piccoli task svolti ciscuno da un singolo sottoprogramma.
Ad esempio funzione che dice se numero è primo o meno, retituisce '1' se primo '0' se non primo. Ad esempio anche 'scanf()', 'print()' e 'gets()' sono sottoprogrammi.

Sottoprogramma non viasualizza, calcola ed elabora sulle variabili.
Sottoprogramma non chiede direttamente all'untente, riceve informazioni dal main (A meno che non si acquisice valori matrice).
Come blackbox (immagine gg) che può retituire informazione come risultato dell'elaborazione (massimo 1) o può solo gestire informazioni senza return (minimo 0). Se produce più di un informazione, gestire un input in n parti diverse per non ripetere ciclo n volte (prossima lezione).

Sintassi funzione base:

```c
/* Sintassi funzione base */
tipo_funzione nome_funzione (argomenti scambiati){
        codice sottoprogramma 1;
        codice sottoprogramma 2;
        codice sottoprogramma 3;
        return dato_restituito;
}
```

Tipi di funzione in base a tipo di dato retituito (anche nuovo tipo con 'typedef struct'), possono essere:

- int
- float
- char
- void (se non entra nulla in input) es: $'void nome_funzione(){elaborazione}'$ o anche $'nome_funzione(){elaborazione}'$

Esempio di funzione 'void' per input valore:

```c
/* Sintassi funzione void */
void menu(){
        printf("----------------------\n");
        printf("inserisci valore\n");
        printf("0, termina\n\n");
        return; /* inutile perche funzione void */
}

int scelta_menu(){
        int sell;
        do{
                /* visualizza menù */
                menu();
                scanf("%d",&sell);
        }while(sell<0 || sell>4);         /* o anche (!(sell>=0 && sell<=4)) */
        return sell;
}

int primo(int val){ /* '1' primo - '0' non primo */
        int i, ris, meta;
```

```
        if(val>0){                /* controllo positivita numero */
                ris = 1;
                if(val%2 == 0){
                        ris = 0;
                }
                else{
                        meta = val/2;                /* tanto 20 non è divisibile per 'i>10' sicuro */
                        for(i=3; i<meta && ris != 0; i += 2){
                                if(val%i == 0)
                                        ris = 0;
                        }
                }
        }
        else
                ris = -1;
        return ris;
}
```

return torna alla funzione main e interrompendo flusso codice della funzione void, non metterne due di return. Al limite mettere 'if(cond) return 0; return1;'
Convenzione disposizione delle parti del codice:

```
/* Distribuzione delle parti di programma */
#include <librerie>
#define DEFINE
/* prototipo */
void mem()
int scelta_menu();
int primo();
int main(int argc; cher*argv[]){

        /* programma del main */

}
```

Passaggio dei parametri con i sottoprogrammi, riassunto generale con parametri necessari:

```
int lunghezza(char[]);
int index_max(int[],int);
int visualizz_matt(int[][NCOL], int , int);
```

È importante sottolineare che le varibili in C hanno uno scope, ovvero sono definite solo all'interno della funzione in cui sono state dichiarate, le varibili dichiarate nella funzione main sono dette variabili globali.

Esempi:

- Funzione che riceve in ingresso una stringa e retituisce lunghezza della stringa (dimesione stringa):

```
/*  Funzione che riceve in ingresso una stringa e retituisce lunghezza della stringa (dimesione stringa) */
int len_stringa(char s[]){
        int dim;
        for(dim=0; s[dim]!='\0'; dim++);
        return dim;
}
```

- Funzione che riceve in ingresso un array e retituisce indice dell'elelmento di valore massimo (si può usare anche 'strlen()' includendo la libreria 'string.h':):

```c
/* retituisce indice dell'elemento dell'array di valore massimo (se 2 pari tengo il più basso di indice) */
int index_max(int v[], int dim){
        int i, i_max;
        i_max = 0;
        for(i=0; i<dim; i++)
                if(v[i]>v[i_max])
                        i_max = i;
        return i_max;
}
```

- Array bidimensionale di valori interi e paramentro strettamente necessrio visuslizza il contenuto:

```c
/* array bidimensionale di valori interi e paramentro
strettamente necessrio visuslizza il contenuto */
void visualizz_mat(int m[][nc]; int nr, int nc) /* [ necessario specificzre tutte le dimensioni dello spazio per linerizzazzi
        int, i j;
        for(i=0; i<nr; i++){
                for(j=0; j<nc; j++)
                        printf("%d";m[i][j]);
        printf(/n);
        }
}
```

- Ingresso stringa e carattere e restituisce ultima posizione in cui quel carattere compare nella string ('-' se non compare):

```c
/* ingresso stringa  e carattere  e restituisce ultima posizioni in cui
quel carattere compare nella string ('-1' se non compare) */
int compare(char stringa[], char caratt){
        int i, pos;
        pos = -1;
        for(i=0; stringa[i]!='\0'; i++){
                if(stringa[i]==caratt)
                        pos=i;
        }
        return pos;
}
```

- Ingresso stringa e carattere e restituisce prima posizione in cui quel carattere compare nella string ('-' se non compare):

```c
/* ingresso stringa  e carattere  e restituisce ultima posizioni in cui
quel carattere compare nella string ('-1' se non compare) */
int compare(char stringa[], char caratt){
        int i, pos;
        pos = -1;
        for(i=0; stringa[i]!='\0' && pos!=-1; i++){
                if(stringa[i]==caratt)
                        pos=i;
        }
        return pos;
}
```

- Esercizio completo 1:

```c
/* trovare carattere in una stringa tramite funzione */
#define LMAX 35
#define TR 'trovato'
#define NT 'non trovato'
int cerca(char[],char)
int main(int argc, char*argv[]){
        char voc[LMAX+1], sel;
        int ris;
        scanf("%s %c", voc, &sel);
        ris = cerca(voc, sel);
        if(ris!=-1)
                printf("%s\n",TR);
        else
                printf("%s\n",NTR);
        return 0;
}
```

- Esercizio completo 2:

```c
/* chiede all'untente numero di valori da getstire e avvalemndosi del sottoprogramma max_array visualizza il massimo */
#include <stdio.h>
#define MAX 50
max_array(int [], int);
int main(int argc, char*argv[}){
        int valori[50], n_val, i;
        do
                scanf("%d", &n_val);
        while(!(n_val>=1 && val<=50));
        for(i=0;i<n_val; i++)
                scanf("%d", &valori[i]);
        pos = max_array(valori, n_val);
        max = valori[pos];
        printf("%d", max);
        return 0;
}
```

5.3.2 Puntatori

Con la sintassi '&nome-variabile' si ottiene la locazione nella memoria nel quale è salvato quel tale valore. Il programma passa, ad esempio, valori array[], locazione memoria del primo output e locazione di memoria del secondo output. La funzione associerà a quel indirizzo di locazione nella memoria il valore risultato dell'elaborazione. tipi di indirizzi:

- 'int* nome-varibile' è l'indirizzo di una generica varibile intera.
- 'float* nome-varibile' è l'indirizzo di una generica varibile reale.
- 'char* nome-varibile' è l'indirizzo di una generica varibile carattere.

Esempio sintassi di funzioni con output mandato tramite puntatore:

```c
/* sottoprogramma che riceve n dati in ingresso e trasmette (non restituisce con return) m>=2 caratteri */
/* ricevuto in ingresso una stringa, trasmette al chimante numero vocali e nuemero consonanti  */
void conta_voc_cons(char s[], int* n_vocali, int* n_consonanti){
        int i;
        int n_voc, n_cons;
        n_voc = 0;
        n_cons = 0;
```

```
        for(i=0; i!='\0'; i++){
                if(s[i]=='e'|| s[i]=='e'|| s[i]=='e'|| s[i]=='e'|| s[i]=='e')
                        n_voc++;
                else
                        n_cons++;
        }
        *n_vocali = n_voc;
        *n_consonanti = n_cons;
}
/* per chiamare questa funzione: */
char voc[LMAX+1];
int num_vocali, num_consonanti;
gets(voc);
conta_voc_cons(voc, &num_vocali, &num_consonanti)
printf("%d\n%d\n",num_vocali, num_consonanti);
```

Per non fare confusione fra l'utilizzo di '*' e '&':

- per trasmettere l'indirizzo di una variabile '*' (quando scrivo la funzione)
- Per chiamare la funzione di utilizza '&'
- asterisco '*': funzione che restituisce il valore della varibile in un indirizzo.
- e commerciale '&': funzione che retituisce indirizzo in cui è allocata una variabile.

Record di attivazione:
Viene creato uno stack (pila) che contine la memoria generata dalla funzione. Ci sono alcune celle che sono dedicate alla variabili passate alla funzione che vengono salvate in locale, ma è molto importnate sapere l'indirizzo di ritorno. Ad esempio quando viene eseguita funzione main c'è solo quelli, chiamata la funzione scanf si crea il record di attivazione del sottoprogramma che poi viene tolta andando all'indirizzo di ritorno.

Passaggio di struct nelle funzioni:

```
/* Distribuzione del codice */
#include
#define
typedef                 ;
/* prototipi        ;        */
int main(){
        /* codice del main */
}
int funzione(){
        /* istruzione della funzione */
}
```

Sintassi del passaggio di una struttura dati di tipo 'struct':

```
/* Sintassi base del passaggio per indirizzo di una struct */
typedef struct data_s{
        int giorno, max;
        int anno;
}data_t;

data_t a, b;

```

```
ris = f1(a,b);
ris = f2(&a,&b);
```

Esercizio con passaggio di struttre a funzioni per indirizzo:

```
/* sottoprogramma che ricevuti in ingresso due date 'da' e 'a' restituisce il numero di giorni tra le due date */
#include <stdio.h>
typedef struct data_s{
        int giorno, max;
        int anno;
}data_t;
int diff_date(data_t *da, data_t *a){
        int diff, m;
        if(*da.anno == *a.anno)                /* *a.anno corripsonde a->anno */
                if(*da.mese == *a.mese)
                        diff = *a.giorno - *da.mese;


        return differenza;
}
int giornidelmese(int mese; int anno){
}

```

Una notazione alternativa per scrivere i puntatori di strutture dati complesse (tipo 'struct') possimao scrivere al posto di '*a.anno' come 'a– anno'.

Metodi alternativi per il passaggio di parametri a funsioni:

- Array:
  function(int v[ ], int dim)
  function(int *v, int dim)
- Matrice: function(int m[ ][NCOL], int nr, int nc)
  function(int *m, int nr, int nc, int ncdmax) // poi gestitre tutto con puntatori per centrare la cella giusta
- Struttura:
  function(mio-tipo-t *s)

5.3.3 Acquisizione da riga di comando

Serve per creare terminali di lavoro con poca iterazione necessaria.
Per compilare e creare l'eseguibile si scrive 'gcc -o nome-programma nome-programma.c'. 'argv[ ]' è un vettore di stringhe (si vede da 'char *argv[ ]') composto da '/.nome-programma ciao 55 volte'.
Essendo un array non si ha i terminatore '\0', quindi si deve fornire anche il numero di elementi 'int argc' (cardinalità). Per convertire i vari 'argv[2], argv[3], ...':

- 'atoi' : converte stringa di 'argv[]' in integer.
- 'atof' : converte stringa di 'argv[]' in float.

Esempio 1 di acquisizione da riga di comando (non c'è 'scanf()'):

```
/* main per funzione  fattoriale_r*/
int main(argc, char *argv[]){
        int val, ris;
        char *numero;
        if(argc==2){
                numero = argv[1];
```

```
                val = atoi(numero);
                ris = fattoriale_r(val);
                printf("%d\n", ris);
        }
        else
                printf("!!!        parametri errati 2!!!\n");
        }

}
```

Esempio 2 di acquisizione da riga di comando:

```
/* main per funzione  */
int main(argc, char *argv[]){
        int val, ris;
        char *seq;
        if(argc==3){
                seq = argv[1];
                c_str = argv[2];
                c = cstr[0];
                ris = verifica_presenza(seq, c);
                printf("%d\n", ris);
        }
        else
                printf("!!!        parametri errati 2!!!\n");
        }
}
```

5.3.4 Contatori

Contatori: sono variabili come i, j che servono per tenere il conto del numero di iterazoni svolte, siccome sono operazioni utilizzate frequentemente, si può possono scrivere nella forma:

- Operatore di incremento: $num = num + 1; \implies num++;$
- Operatore di decremento: $num = num - 1; \implies num--;$

Gli operatori di incremento e decremento si possono scrivere anche nella forma ++num; e –num, ma noi non useremo il postincremento.
Si possono abbreviare anche le operazioni in questo modo:

- Somma: $tot = tot + cont \implies tot+ = cont$
- Differenza: $tot = tot - cont \implies tot- = cont$
- Prodotto: $ris = ris * val \implies ris* = val$
- Divisione: $ris = ris/val \implies ris/ = val$

5.3.5 Casting

L'operazione di casting ha la sintassi '(tipo_variabile)nome_variabile_castare'. È importante sottolineare che tronca la visualizzazione della variabile, non fa cambiare il valore di essa, infatti bisogna assegnarlo per salvarlo in un'altra variabile.
La'operazione ha priorità maggiore ripetto agli operatori classici, ad esempio scrivendo 'avg=(float)tot/num' viene svolto prima casting di 'tot' a reale e poi viene svolta la divisione fra reali, mentre per svolgere prima la divisione si scrive 'avg=(float)(tot/num)'.
Notiamo che spesso questa operazione è ridondante perchè se assegnamo ad un intero (int) un valore di tipo reale (float), esso viene troncato perchè non ci sono abbastanza bit per rappresentarlo.

```c
/* Cambiare tipo di una variabile */
int main(int argc, char*argv[])
{
        float val;
        int dif;
        scanf("%f",&val);
        dif=(int)val;         /*cast esplicito*/
        print("%d\n", dif)
}
```

5.4 Costrutti in C

```c
/* Elenco di tutta la sintassi da imparare a memoria */
#include <_nome_>   #define SYMB VAL
typedef _definizione_ _nomet_;
typedef struct _nomes_ _definizione_ _nomet_;
int main(int argc, char * argv[])   return   /* */   ;   ,
_tipor_ _nomesottop_(_tipoin_ _nomep_, _tipoin_ _nomep_);
sizeof   (_tipo_)   ()   []   {}
void   int _nomev_;   char _nomev_;   float _nomev_;
FILE * _nomev_;   _nomet_ _nomev_;   ''   '\"   '\0'
=   +   -   *   /   %   ++   --   +=   -=   *=   /=
&&   ||   !   ==   !=   >   >=   <   <=   ->   .   &   *
if(_espr_) _istr_   if(_espr_) { _istr_ _istr_ }
if(_espr_) _istr_ else _istr_
if(_espr_) _istr_ else if (_espr_) _istr_ else _istr_
if(_espr_){ if(_espr_) } else _istr_
switch(_espr_){ case _caso_: _istr_ case _caso_: _istr_
default: _istr_ }   break
while(_espr_) _istr_   while(_espr_){ _istr_ _istr_}
do _istr_ while(_espr_);   do{ _istr_ _istr_} while(_espr_);
for(_espr_;_espr_;_espr_) _istr_
scanf   printf   gets   fopen   fclose   feof   EOF   fscanf
fprintf   fgets   fread   fwrite   malloc   free   atoi
strlen   strcmp   strcpy
```

5.4.1 Costrutto if

```c
/* Sintassi if base */
if(espressione)
    istruzione_Vero;
```

Per scrivere 3 espressioni nell'if bisogna utilizzare le parentesi graffe per riunire costrutti che vanno svolti dopo una condizione, senza graffe solo l'espressione successiva viene eseguita (l'indentazione è solo un aspetto visivo).

```c
/* Sintassi if con più istruzioni*/
if(espressione){
    istruzione1_Vero;
    istruzione2_Vero;
    istruzione3_Vero;
}
```

Programma valore assoluto con cotrutto if:

```c
/* Valore assoluto */
int main(int argc, char*argv[]){
    float val;
    scanf("%f", &val);
    if(val < 0}
        val=-val;
    printf("\d", val);
}
```

Programma valore assoluto con cotrutto if else:

```c
/* Valore assoluto salvando dato iniziale */
int main(int argc, char*argv[]){
    float val;
    scanf("%f", &val);
    if(num < 0}
        abs=-val;
    else if(val > 0) /*ulteriore condizione ridondante, anche solo: else operazione*/
        abs=val;
    else
        operazione
    printf("\d", abs);
}
```

Se una istruzione si trova sia nell'if che nell'else significa che si svolge sempre, questa istruzione è ridondante ed è consigliabile metterla prima dell'if in modo da scriverla una volta sola.

Programma che acquisisce valore input, se negativo restituisce '-', se positivo '+' e se nulla uno spazio ' ' (if-else-annidato):

```c
/* Retituisce simbolo in base al segno del valore */
#define POS '+'
#define NEG '-'
#define NUL ' '
int main(int argc, char*argv[]){
    int val;
    char ris;
    scanf("%d", &val);
    if(val>0)
        ris = POS;
    else
        if(val<0)
            ris = NEG;
        else
            ris = NUL;
    printf("%c",ris);
}
```

In alcuni linguaggi come C, C++ e Python si può scrive nella stessa riga anche 'if else' senza indentare la condizione rispetto al 'if' principale:

```c
/* Sintassi if ad elenco */
if(espressione)
    istruzione1_Vero;
else if(espressione)
    istruzione2_vero;
else if(espressione)
    istruzione3_vero;
```

```
else intruzione4_finale;
```

È importante sottolineare che else è legato al precedente if che non ha già un altro else legato (non possono esseci due else per lo stesso if, nel caso si deve creare un if annidato che fa parte del'else del primo if delimitato da graffe).

All'interno dell'if si valutano le condizioni, ma di fatto sono espressioni che retituiscono un booleano '0'(F falso) o '1' (T vero); queste espressioni possono essere combinate tramite operatori logici e il risultato si controlla con le tavole della verità degli operatori logici.

Ad esempio possiamo scrivere 'if(val==0)' per svolgere delle operazioni solo nel caso in cui è vera l'espressione val==0, che è uguale a 'if(!val)'.
La condizione opposta sarebbe 'if(val!=)', che è uguale a 'if(val)'.
Per confrontare con una costante possiamo scrivere 'if(5==val)', così se ci dimentichiamo '==' e mettiamo solo '=' il compilatore segnala errore di sintassi.

5.4.2 Switch

Costrutto di selezione particolare che selezione in base al valore di una variabile il tipo di istruzioni da eseguire. Possiamo usarlo quando abbiamo una variabile di tipo intero o che sottende un intero (quindi anche con char).

Sintassi del contrutto 'switch()':

```
switch(var){
        case 1: istruzione_1;        /* con char:        case 'a': .... */
                        break;        /* necessario per uscire (altrimenti esegue tutto) */
        case 2: istruzione_2;

        default: intruzione_n;        /* facoltativo */
}

```

5.4.3 Ciclo While e Do-while

Costrutto che permette di itereare delle operazioni fin tanto che (in inglese while) una condizione è vera, quando è falsa si esce e continua il flusso lineare del codice. Usando il While la condizione viene posta prima di svolgere le operazioni, mentre usando il Do-while la condizione di uscita viene posta dopo aver svolto le operazioni.
Nel costrutto While è necessario che vengano svolte delle istruzioni che modificano la variabile nela condizione del ciclo, in modo da evitare i loop infinti.

```
/* Sintassi while */
while (espressione)
    istruzione_Vero;
```

Il costrutto while pone la condizione dopo aver svolto una volta tutte le istruzioni, è utili anda esempio per controlla re che il valore inserito in input in una variabile sia accettabile.

```
/* Sintassi while */
Do
    istruzione;
while (espressione);
```

Calcola valore massimo (While):

```c
/* programma che acquisisce 20 valori interi e acquisisce valore massimo */
#include <stdio.h>
#define N_CICLO 20
int main(int argc,char*argv[]){
        int cont=1,num,max;
        scanf("%d\n",&max);
        while(cont<N_CICLO){
                scanf("%d\n",&num);
                if(max<num)
                        max=num;
                cont++;
        }
        printf("%d\n",max);
        return 0;
}
```

Ricevere dati in input fin che non si inserisce numero specifico (While):

```c
/* programma che acquisisce primo valore intero 'fine' seguito da una sequenza di valori interi
a priori di lunghezza ignota che si ritiene termianta quando l'utente inserisce tale valore
massimo minimo e media dei valori, se subito valore fine visualizza max min e medi 27*/
#include<stdio.h>
int main(int argc,char*argv[]){
        int stop, val, max, min, tot;
        float avg, count;
        scanf("%d\n%d",&stop,&val);
        max=val;
        min=val;
        count=0;
        while(stop!=val){
                scanf("%d",&val);
                count++;
                tot+=val;
                if(max<val)
                        max=val;
                else if(min>val)
                        min=val;
        }
        if(min!=stop)
                avg=(float)tot/count;
        else
                avg=stop;
        printf("massimo: %d\nminimo: %d\nmedia: %f\n", max, min, avg);
        return 0;
}
```

Acquisire numero positivo e calcolarne numero cifre (Do-while):

```c
/* chiedere all'utente valore fin che non è strettamente positivo, poi calcolare numero di cifre da cui è composto */
#include<stdio.h>
#define BASE 10
int main(int argc,char*argv[]){
        int val,cifre,count;
        do{
        scanf("%d",&val);
        }while(val<=0); // input numero fin che non è positivo
        count=val; // così salvo valore iniziale
        cifre=0;
```

```
        do{
                count/=BASE; // divido numero per 10
                cifre++; // incremento il contatore delle cifre
        }while(count>0);// fin che il numero non diventa minore di zero (num>(int)0.999=0)
        printf("%d\n",cifre);
        return 0;
}
```

5.4.4 Costrutto for

Il costrutto for è extra e facoltativo, ma utile per fare iterazioni a conteggio, in particolare è utile per getisre dati salvati nin array.
Il for é un ciclo a condizione iniziale, come while con contatore che fa finire il ciclo dopo un numero definito a priori di iterazioni.

La sintassi (pseudocodice) del ciclo for è la seguente:

```
/* Sintassi pseudocodice ciclo for */
#define numero delle iterazioni
int contatore;
for(inizio contatore; condizione di uscita; incremento contatore)
        operazioni ed elaboraizoni
```

La sintassi (base) del ciclo for è la seguente:

```
/* Sintassi base ciclo for */
#define ITERAZIONI numero_iterazioni
int contatore;
for(contatore=0; contatore<ITERAZIONI; contatore++)
        operazioni ed elaboraizoni
```

La sintassi (completa) del ciclo for è la seguente:

```
/* Sintassi condizioni multiple ciclo for */
#define ITERAZIONI numero_iterazioni
int contatore;
for(count1=0, count2=0; condiz1 && condiz2 || condiz3; count1++, count2--){
        operazione1;
        operazione2;
        operazione3;
}
```

Lo scopo del for è scrivere in maniera più compatta un particolare ciclo while che fa scorrere l'array. Non è da usare sempre il for al posto del while, al limite aggiungere condizione di uscita dal ciclo for nella sezione delle condizioni oltre al fatto di non scorrere array otre la sua dimensione.
Non si devono dichiarare variabili locali che valgono solo nel for, meglio usare solo variabili globali. Considerare che le varibili globali usate nel for non si azzerano e il valore non cambia.

Esercizi con il ciclo for:
Conversione da decimale a binario (configuarazione con tutti i bit):

```
/* programma che acquisito valore natirale 1<=num<=1023 e fin che non è tale lo richiedo,
programma calcola e visualizza la sua rappresentazione nel sistema binario (tutti i bit) */
#include <stdio.h>
#define BASE 2
#define MIN 1
#define MAX 1023
```

```c
#define BIT 10 /* formula configuazioni bit BIT=log_BASE(MAX-MIN)=10 */
int main(int argc, char*argv[]){
        int num, binario[BIT], resto, i;
        do
                scanf("%d",&num);
        while(!(num>=MIN && num<=MAX)); /* anche (num<MIN || num>MAX)*/
        resto = num;
        for(i=0; i<BIT; i++){
                binario[i] = resto % BASE;
                resto = resto / BASE;
        }
        for(i=BIT-1; i>=0; i--){
                printf("%d",binario[i]);
        }
        printf("\n");
        return 0;
}
```

Conversione da decimale a binario (configuarazione con solo i bit necessari):

```c
/* programma che acquisito valore natirale 1<=num<=1023 e fin che non è tale lo richiedo,
programma calcola e visualizza la sua rappresentazione nel sistema binario (minimo bit) */
#include <stdio.h>
#define BASE 2
#define MIN 1
#define MAX 1023
#define BIT 10 /* formula configuazioni bit BIT=log_BASE(MAX-MIN)=10 */
int main(int argc, char*argv[]){
        int num, binario[BIT], resto, i;
        do
                scanf("%d",&num);
        while(!(num>=MIN && num<=MAX)); /* anche (num<MIN || num>MAX)*/
        resto = num;
        i = BIT-1;
        while(resto>0){ /* non usare for perchè ciclo a condizione e non conteggio */
                binario[i] = resto % BASE;
                resto = resto / BASE;
                i--;
        }
        for(i++; i<BIT; i++){
                printf("%d",binario[i]);
        }
        printf("\n");
        return 0;
}
```

Conversione da decimale a binario di due numeri (configuarazione a bit necessari del numero maggiore):

```c
/* programma che acquisito valore natirale 1<=num<=1023 e fin che non è tale lo richiedo, programma calcola e
visualizza  a sua rappresentazione nel sistema binario2 usando lo stesso numero di cifre (minimo indispensabile) */
#include <stdio.h>
#define BASE 2
#define MIN 1
#define MAX 1023
#define BIT 10 /* formula configuazioni bit BIT=log_BASE(MAX-MIN)=10 */
int main(int argc, char*argv[]){
        int num1, num2 , binario1[BIT], binario2[BIT], i, supp, k;
        do
                scanf("%d",&num1);
        while(!(num1>=MIN && num1<=MAX)); /* anche (num<MIN || num>MAX)*/
```

```
        do
                scanf("%d",&num2);
        while(!(num2>=MIN && num2<=MAX));
        if(num2>num1){
                supp = num1;
                num1 = num2;
                num2 = supp;
        }
/* numero in binario 1 */
        i = BIT-1;
        while(num1>0){
                binario1[i] = num1 % BASE;
                num1 = num1 / BASE;
                binario2[i] = num2 % BASE;
                num2 = num2 / BASE;
                i--;
        }
        k = i;
/* output */
        for(i++; i<BIT; i++){
                if(num1>=num2)
                        printf("%d",binario1[i]);
                else
                        printf("%d",binario2[i]);
        }
        printf("\n");
        for(k++; k<BIT; k++){
                if(!(num1>=num2))
                        printf("%d",binario1[k]);
                else
                        printf("%d",binario2[k]);
        }
        printf("\n");
        return 0;
}
```

5.4.5 Ricorsione

Metodo particolare di risolvere un problema. Scrivere un sottoprogramma che, o direttmente o indirettamente, richiama se stesso.

```
// Ricorsione diretta
___f(...){
                istruzioni;
}

___g(...){
                istruzioni;
}

// Ricorsione indiretta
___f(...){
        ___g(...){
                istruzioni;
        }
}

___g(...){
        ___f(...){
```

```
                istruzioni;
        }
}
```

Sottoprogramma di esempio (fattoriale) del metodo ricorsivo:

```
// ricorsione fattoriale:       metodo 1
int fattoriale(int num){
        int ris;
        if(num==0 || num==1)
                ris = 1;
        else
                ris = num * fattoriale(num-1);
        return ris;
}
```

// ricorsione fattoriale: metodo 2 (compatto)

```
int fattoriale(int num){
        if(num==0 || num==1)
                return 1;
        return num * fattoriale(num-1);
}
```

Esempio di sottoprogramma ricorsivo:

1. 'strlen()':

```
                /* sottoprogramma ricorsivo che ricevuta in ingreso una stringa,
                conta e restituisce al chiamante la lunghezza della stringa */
                strlen_r(char stringa[]){
                        if(stringa[0]=='\0')
                                return 0;
                        return 1+strlen_r((&stringa[0])+1);
                }
```

2. Contare ripetizioni carattere:

```
                        /* sottoprogramma che ricevuto in ingresso stringa e carattere conta quante volte è contenuto */
                        conta_ripetizioni(char stringa[], char carattere){
                                int count;
                                if(stringa[0]=='\0')
                                        count = 0;
                                if(s[0]==carattere)
                                        count = 1;
                                else
                                        count = 0;
                                count += conta_ripetizioni(&stringa[1], c);
                                return count;
                        }
```

3. Controllare se compare carattere in una stringa:

```
                        /* sottoprogramma che ricevuto in ingresso stringa e carattere verifica se compare o meno */
                        verifica_presenza(char stringa[], char carattere){
                                int count;
```

```
                if(stringa[0]=='\0')
                        return 0;
                else if(s[0]==carattere)
                        return 1;
                else
                        return verifica_presenza(&stringa[1], c);
        }
```

5.5 Strutture dati

5.5.1 Array monodimensionali

gli array monodimensionali sono anche detti vettori, ad esempio $\vec{v}[5] = (10, 4, 3, 2, 1)$
Per ora abbiamo gestito situazioni e problemi dove non era necessario salvare i valori di cicli indefiniti, i dati in se li abbiamo cancellati e salvavamo i risultati parziali delle iterazione in delle variabili.

Se dobbiamo salvare 50 valori non è comodo utilizzare 50 variabili, quindi introduciamo una nuova struttura dati detta array che può avere cardinalità definita dal programmatore. La variabile è un array[1][1]

Definition 5.1 (Array)**.** *è qualle struttra dati (variabile strutturata) che ci permette di definire una cardinalita con un unico nome che stabilisco a priori che conterra un numero definito di valori.*

Un array contiene dati omogenei, tutti interi o float ecc, e questa quantità è costante e nota a priori. Nella fase di compilazione deve essere nota a priori, nello standard ANSI non usiamo array dinamici (faremo alloracizone dinamica malloc).

In linguaggio C si parte da elemento 0, quindi se abbaimo un array di dimensione 10 il primo valore sarà in posizione 0 e l'ultimo in posizione 9. Sintassi dichiarazione e utilizzo degli array:

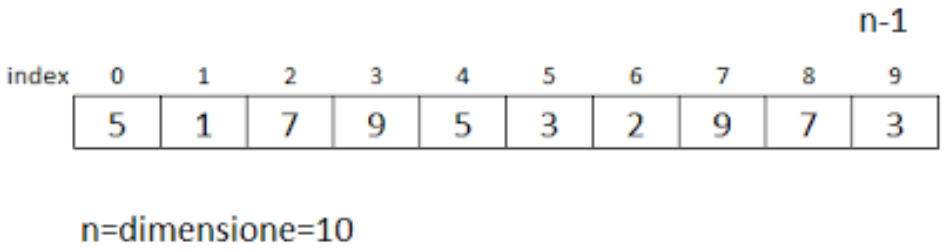

Figure 18: Array monodimesionali

```
/* Sintassi array monodimensionale */
#define NUM_ELEMENTI numero elementi
tipo_array nome_array[NUM_ELEMENTI];
int numeri[NUM_ELEMENTI];
float soldi[30];
```

Per accedere agli elementi si utilizza ciclo e variabili contatatori (es: i, j, h, k).
Ciclo while che fa scorrere elementi in input, sintassi per popolazione di array monodimensionale:

```
/* Sintassi popolazione array monodimensionale */
int numeri[num_elementi]
while(i<num_elementi){
        scanf("%d",&numeri[i])
}
```

Esercizio di esempio, calcolo della media dei valori inseriti e printa valori dell'array superiori alla media:

```c
/* Programma che acquisice 50 valori interi e visualizza valori supriori al valore medio */
#include <stdio.h>
#define CELLE 5
int main(int argc,char*argv[]){
        int i, valore[CELLE],tot;
        float avg;
        i = 0;
        tot = 0;
        while(i<CELLE){
                scanf("%d",&valore[i]);
                tot += valore[i];
                i++;
        }
        avg = (float)tot / CELLE;
        i = 0;
        while(i<CELLE){
        if(valore[i]>avg)
                printf("%d\t",valore[i]);
        i++;
        }
        printf("\n");
        return 0;
}
```

5.5.2 Array multidimensionali

Struttura dati oraganizzata su due dimensioni (es: immagine, tabella). Se si devono rappresentare una lista di 50

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Figure 19: Array multidimensionali in C

numeri si usano array monodimensionali, mentre array multidimensionali si usano per le matrici; qui riporto la sintassi base di un array multidimensionale:

```c
/* Sintassi array multidimensionale base */
tipo_array nome_array[num_righe][num_colonne];
int mat[N_RIGHE][N_COLONNE];
```

Si usano le varibili indice 'i' e 'j' che rappresentano rispettivamente il nuemro di colonne e il numero di righe. Dichiarazione e popolazione di un array multidimensionale:

```c
/* Sintassi popolazione array multidimensionale */
#define N_RIGHE 8
#define N_COLONNE 5
int mat[N_RIGHE][N_COLONNE], i, j;
for(i=0;i<N_RIGHE;i++){
        for(j=0;j<N_COLONNE;j++){
                scanf("%d",mat[i][j]);
        }
}
```

Esercizio con array bidimensionale:

```c
/* programma che acquisisce dati array buismensionele quadrato di dimensione 5
programma calcola e visulaiiza 1 se la matrice è identità */
#include <stdio.h>
#define DIM 5
int main(int argc,char*argv[]){
        int matrice[DIM][DIM], i, j, ris;
        for(i=0;i<DIM;i++){ /* Input matrice */
                for(j=0;j<DIM;j++){
                        printf("riga %d, colonna %d: ",i+1,j+1);
                        scanf("%d",&matrice[i][j]);
                }
        }
        ris = 1;
        for(i=0;i<DIM && ris != 0;i++){ /* Iscorrimento righe */
                for(j=0;j<DIM && ris != 0;j++){ /* scorrimento colonne */
                        /* controllo  se diganoale è 1 e resto 0*/
                        if((i==j && matrice[i][j]!=1) || (i!=j && matrice[i][j]!=0))
                                ris = 0;
                }
        }
        printf("%d\n",ris);
        return 0;
}
```

.

Linearizzazione della memoria:
la memoria è monodimensionale (vettore-colonna), di conseguenza bisogna portare in forma monodimensionale un array bidimensionale. Di per sé si potrebbe gestire una matrice multidimensioanle come un vettore, ma risulterebbe complicato nellla lettura da parte di un altro utente. Esercizio con gli array multidimensioneali, verificare se un

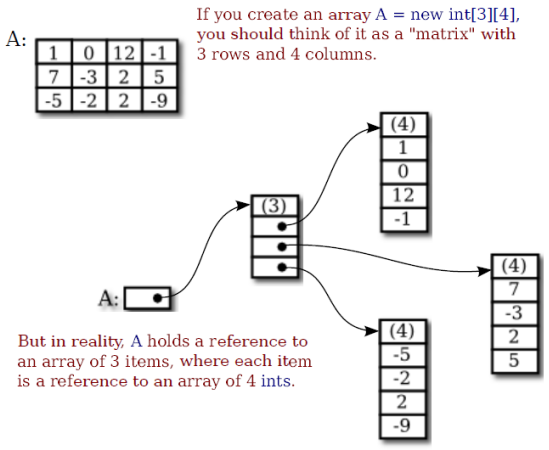

Figure 20: Linearizzazione della memoria

quadrato è magico (vedere se la somma delle righe, colonne e diagonali è uguale)

```c
/* verificare s è quadrato magico (somma righe / colonna / digonali è uguale) output si '1' // no '0' */
#include <stdio.h>
#define DIM 3
int main(int argc, char*argv[]){
        int matrice[DIM][DIM], i, j, mag, sum1, sum2, sum3;
        mag = 1;
```

```
        sum1 = 0;
        sum2 = 0;
        sum3 = 0;
        for(i=0;i<DIM;i++){ /* Input matrice */
                for(j=0;j<DIM;j++){
                        printf("riga %d, colonna %d: ",i+1,j+1);
                        scanf("%d",&matrice[i][j]);
                }
        }
        for(i=0;i<DIM;i++) /* scorrimento digonale principale */
                        sum1 += matrice[i][i];
        for(i=0;i<DIM;i++) /* scorrimento digonale secondaria */
                        sum2 += matrice[i][DIM-i-1];
        if(sum1!=sum2)
                mag = 0;
        for(i=0;i<DIM && mag!=0;i++){ /* scorrimento esterno righe/colonne */
                sum1 = 0;
                sum3 = 0;
                for(j=0;j<DIM && mag!=0;j++){
                        sum1 += matrice[i][j];
                        sum3 += matrice[j][i];
                }
                if((sum1!=sum2) || (sum3!=sum2))
                                mag = 0;
        }
        printf("%d %d\n",mag,sum1);
        return 0;
}
```

5.5.3 Structure

Salvare dati in struttura dati come array ma senza vincolo del tipo, struct possono contenere diverse tipologie di variabili, array mono e multidimensionali, il tutto gestito come unica identità e campi di tipologia diversa (La typedef va messa prima del main).

Sintassi del costrutto struct:

```
/* Sintassi base struct */
struct nome_struttura_s{
        tipo1 nome1;
        tipo2 nome2;
}
```

Esempio di utilizzo:

```
/* Sintassi esempio 1 struct */
struct data_s{
        int giorno, mese;
        float soldi;
}
```

Esempio di struct composto da variabili e array di tipo diverso:

```
/* Sintassi esempio 2 struct */
struct studente_s{
        char cognome[MAX_LEN+1];
        char nome[MAX_LEN+1];
        int erasmus;
```

```
        bool laureato;
        float media;
}
```

Per definire un nuovo tipo di struct e poterlo richiamare per utilizzarlo più volte nel programma si deve definire il tipo di struttra dati con il comando '*typedef*' (non è necessario sia dichiare struttura e definire nuovo tipo):

```
/* Sintassi typedef struct */
typedef struct nome_struttura{
        tipo1 nome1;
        tipo2 nome2;
}nome_struttura_t;
```

Esempio di utilizzo del comendo '*typedef*':

```
/* Sintassi typedef esempio struct */
typedef struct data_s{
        int giorno;
        int mese;
        int anno;
}data_t;

data_t nascita_s;
nascita_s.giorno = 13; /* assegnamento mirato */
nascita_s.mese = 03;
scanf("%d",&nascita.anno);
```

Confronto fra date:

```
/* acquisire 50 date di nascita e vedere quanti sono nati nello stesso giorno */
#include <stdio.h>
#define NUM 50
typedef struct data_s{
        int giorno, mese, anno;
}data_t;

int main(int argc,char*argv[]){
        int i, nascita, comp;
        data_t persone_s[NUM], data_rif_s;
        for(i=0;i<NUM;i++)
                scanf("%d %d %d", &persone_s[i].giorno, &persone_s[i].mese, &persone_s[i].anno);
        scanf("%d %d %d",&data_rif_s.giorno, &data_rif_s.mese, &data_rif_s.anno);
        nascita = 0;
        comp = 0;
        for(i=0; i<NUM; i++){
                if(persone_s[i].mese==data_rif_s[i].mese)
                        if(persone_s[i].giorno==data_rif_s[i].giorno){
                                comp++;
                                if(persone_s[i].anno==data_rif_s[i].anno)
                                        nascita++;
                        }
        }
        printf("%d\t%d\n");
        return 0;
}
```

5.5.4 Strighe

Stringa è struttura dati che non fa accedere a singolo carattere di array, si risparmia nella complessità di gestione infatti è presente un terminatore non contenuto nella stringa.
Array di n caratteri corrisponde ad una stringa di $n+1$ caratteri perchè è presente un carattere che per definizione fa da carattere terminatore, in c è presente il carrattere non visibile '\0'.

Sintassi gestione stringhe:

```c
/* Sintassi stringhe base */
scanf("%s", &seq[0]);
printf("%s\n", &seq[0]);
```

In un array in c, scrivere '&seq[0]' corrisponde a 'seq', quindi è implicito che si operi sulla posizione 0 del vettore che corrisponde alla cella z(0, 0) della matrice.

Per lo scanf spazio ' ', tab ' ' e a capo '\n', quindi si devono sostituire spazi con underscore o trattini per evitare che si fermi la stringa perchè lo spazio viene visto come terminatore.

Per fare ciò è necessario includere la libreria 'string', che si scrive con '#include ¡string.h¿'
Un altro metodo alternativo allo scanf() è get();

```c
/* Sintassi stringhe con gets */
gets(seq);
gets(&seq[0]);
```

Con 'gets()' i tab e spazi sono considerati caratteri, metre il terminatore è solamente a capo 'È pericolosa perchè rischia di sovrascrivere memoria dopo quella dedidcata all'array, con gets si riceve infatti un warning ma lo ignoriamo perchè l'utente sono inserirà più caratteri rispetto al limite.

Sintassi con define:

```c
/* Esempio programma per vedere sintassi stringhe */
        #define LMAX 100
        int main(int argc, char *argv[]){
                char frase[LMAX+1], mask[LMAX+1];
                gets(frase);
                c = INIZIO + imax;
                i = 0;
                while(frase[i] != '\0'){ /* for(i=0; frase[i]!='\0'; i++) */
                        if(frase[i] == c)
                                mask[i] = SOST;
                        else
                                mask[i] = frase[i];
                        i++;
                }
                mask(i)=frase[i]; /* aggiungere terminatore */
        }
```

Contare numero lettere in una stringa:

```c
/* acquisisce sequenza al più 100 caratteri e terminata con carattere '*'
calcolare il carattere più frequente e visualissa frase in cui ha sostituito
il carattere più frequente con '*' - caratteri minuscoli, spazi e interpunzione
e non cè asterisco per primo e a parità di frequenza carattere
preso con max_frequente à l'ultimo nell'alafabeto*/
#define <stdio.h>
#define MAX 100
#define STOP '*'
#define INIZIO 'a'
```

```c
#define FINE 'z'
int main(int argc, char *argv[]){
        char frase[MAX];
        int dim, i, imax, pos, lettere[MAX];

        /* acquisizione */
        dim = 0;
        scanf("%c", &c);
        while(c != STOP){
                frase[DIM] = c;
                dim++;
                scanf("%c", &c);
        }

        /* contatore e inizializzazione*/
        for(i=0;i<ALFA; i++){
                lettere[i] = c;
        }

        /* analisi primo carattere */
        c = frase[0];
        pos = c - INIZIO;
        lettere[pos] = 1;
        imax = pos;

        /* analisi caratteri rimannenti */
        for(i=1; i<DIM; i++){
                if(frase[i]>=INIZIO && frase[i]<=FINE){
                        pos = frase[i] - INIZIO;
                        lettere[pos]++;
                        if(lettere[pos]>lettere[imax])
                                imax = pos;
                        else if(lettere[pos]==lettere[imax])
                                if(pos>imax)
                                        imax = pos;
                }
        }
        /* sostintuzione */
        c = INIZIO + imax;
        for(i=0; i<dim; i++0){
                if(frase[i] == c)
                        frase[i] = SOST;
                printf("%c", frase[i]);
        }
        printf("\n");
        return 0;
}
```

Stabilire se una stringa è palindroma confrontando elementi array[i] e array[DIM-i-1]:

```c
/* scrivere programma che acquisisce una stringa di al piu 100 carratteri, calcola e visualizza '1' so sono palindrome '0' se no */
#include<stdio.h>
#include<string.h>
#define MAX 100
#define PAL 1
#define NO_PAL 0
int main(int argc, char *argv[]){
        char stringa[MAX+1], ris;
        int i, leng;
        gets(stringa);
        ris = PAL;
```

```
        for(leng=0; stringa[leng]!='\0'; leng++);
        i = 0;
        while(ris==PAL && i<leng){
                if(stringa[i]!=stringa[leng-1-i])
                        ris = NO_PAL;
                i++;
        }
        printf("%d\n", ris);
        return 0;
}
```

Contare numero vocali e consonanti in una stringa:

```
/* programma che conta consonanti e vocali, utente inserisce solo carartteri minuscoli*/
#include<stdio.h>
#include<string.h>
#define MAX 100
#define INIZIO 'a'
#define FINE 'z'
int main(int argc, char *argv[]){
        char seq[MAX+1];
        int voc, con, i;
        voc = 0;
        con = 0;
        gets(seq);
        for(i=0; seq[i]!='\0';i++){
                if(seq[i]>=INIZIO && seq[i]<=FINE)
                        if(seq[i]=='a' || seq[i]=='e' || seq[i]=='i' || seq[i]=='o' ||seq[i]=='u')
                                voc++;
                        else
                                con++;
        }
        printf("voc: %d\t cons: %d\n", voc, con);
        return 0;
}
```

5.5.5 Allocazione dinamica

L'indirizzo che non guarda da nessuna parte ha contenuto 'NULL'. Il tipo della varibile che vado a chiamare è un indirizzo di un intero, che all'inizio del programmma sarà senza contenuto. Dopo la variabile la assegno a dove mi è stata data la memoria.

È necessario includere la libereria '¡stdlib.h¿' e la memoria che sarà occupata dall'allocazione dinamica non è lo stack, ma lib.

Le funzioni per l'allocazione dinamica è la 'memory alloc' abbreviata con 'malloc()' :

```
/* Spiegazione della funzione malloc */
void * malloc(size_t)

/* la funzione 'sizeof()' restituisce una varibile di tipo 'size_t' */
sizeof(int) = "quanto occupa di memoria (varia in base ad architettura del calcolatore)"

var = mallc()
if(var!=NULL){        /* if(var) */
        /* operazioni */
        free(var)               /* void free(void*) */
}
else
```

```
        printf("errore di allocazione \n")

```

Se la memoria si esarisce la malloc restituisce 'NULL'. Quindi devo chiedere memoria, se risultato malloc è diverso da 'NULL' procedo con algoritmo, else probblema di memoria.

Devo fare la free solo se il cpntenuto della memoria non serve più, nel caso dopo la si mette a fine del main. La memoria restituitsa dal programmma allocata dionamicamentie non si cancella, quindi è il mezzo ideale per restituire al chiamante da funzione. Si libera e non èpiù accessibili solo dopo la 'free()'.

Esercizio di esempio con allcoazione dinamica:

```
/* Programma che ordina  */
int main(inr argc, char *argv[]){
        int *v;
        int n, i;
        do
                scanf("%d", &n);
        while(N<=1);
        v = (int*)malloc(n*sizeof(int));
        if(v!=NULL){
                for(i=0; i<n; i++)
                        scanf("%d", v+i);        /* &v[i] */
                bsort(v,n);
                for (i=0, i<n; i++)
                        printf("%d\t", *(v+i));        /* v[i] */
                printf("\n");
                free(v);
        }
        else
                printf("errorre di allocazione %d int", n);
        return 0;
}

```

Esercizio allocazione dinamica con sottoprogramma:

```
/* crea e restituisce una nuova stringa che crea e restituisce le consonanti della stringa di partenza.
Scriviamo iun programma da riga di comanfo che acquisita da riga di comando una stringa,
avvalendosi del sottoprogramma, visualizza le consonanti presenti */

int main(int argc, char *argv[]){
        char *seqin;
        char *cseq;
        if(argc==2){
                seqin = argv[1];
                cseq = consonanti(seqin);
        }
        if(cseq!=NULL){
                printf("%s\n", cseq);
                free(cseq);
        }
        else
                printf("argomenti non coerenti\n");
        return 0;
}

char * consonanti(char s[]){
        char *sc;
        int nc, i, j;
```

```
        nc = 0;
        for(i=0; s[i]!='\0'; i++)
                nc += is_cons(s[i]);
        sc = (char *)malloc((nc+1)*sizeof(char));
        if(sc!=NULL){                /* anche malloc e if uniti (NULL==0000000) */
                J = 0;
                for(i=0; s[i]!='\0'; i++){
                        if(is_cons(s[i])==1){
                                *(sc+j) = s[i];
                                j++;
                        }
                }
                *(sc+j) = '\0';
        }
        else
                printf("errorre di allocazione %d char", nc);
        return sc;
}

int is_cons(char caratt){
        if(isalpha(caratt)){
                if(caratt!= vocali)
                        return 1;
                else
                        return 0;
        else
                return -1;
}
```

5.5.6 Linked list

Struttura dati per memorizzare informazioni senza sapere all'inizio il numero di elementi, si possono poi modificare anche in corso le dimensioni.
Permette di allocare memoria dinamicamente tenendo traccia di dove sono i dati, ed eventualmente una volta non più utili utilizzando 'free()' si libera la memoria.
Si deve creare una struttura dati che contiene il valore di quell'elemento della lista e il puntatore all'elemento successivo.

```
/* struttura base di una lista */
typedef struct element{
        int val;
        int *point;
}nodo;
ilist_t *h NULL;
/* sottoprogrammi sulle liste */
ilist_t * append(ilist_t *, int ){        /* (tipo: testa di lista aggiornato; parametri: lista, valore da aggiungere) */4
ilist_t * insert_in_order()        /* aggiunge in ordine (così non si ordina) */
ilist_t * push()         /* aggiunge in coda */
}
```

Esercizio di esempio sulle liste concatenate:

```
/* chiedere a utenete valore intero sentinella, inserire valori che vuole fin che non inserisce il valore sentinella.
Visualizzare tutti i valori maggiori di quelli acquisiti fino a quel momento */
#include<stdio.h>
#include<stdlib.h>
typedef struct ilist_s{
        int val;
```

```
        struct list_s *next;
}ilist_t;

ilist_t *append(ilist_t *, int);         /* dichiarazione prototipo */

int main(int argc, char *argv[]){
        ilist_t *h = NULL;                /* dichiaro puntatore nullo perche lista vuota */
        ilist_T *elem;
        int stop, num, tot, cont;
        float avg;
        scanf("%d", &stop);
        scanf("%d", &num);
        while(num!=stop){
                h = append(h, num);
                scanf("%d", num);
        }
        tot = 0;
        cont = 0;
        elem = h;
        while(elem!=NULL){
                tot += elem->val;         /* sommo valore elemento */        /* 'elem->val' come  '*elem.val' */
                cont++;                /* mi sposto all'elemento successivo */
                elem = elem->next;                /* 'elem->next' come '*elem.next' */
        }
        if(cont>0){
                avg = (float)tot/cont;
                for(el=h; el!=NULL; el=el->next){               /* 'el!=null' come 'el' */
                        if(el->val>avg)
                                printf("%d\t", el->val);
                        else
                                printf("lista vuota");
                        freelist(h);        /* programma che libera la lista come 'free(testa_lista)' */
                }
        }
}
```

Esercizio 2 (versione 1):

```
/* sottoprogramma che riceve array e retituisce tutti e soli i numeri primi nell'array */
typedef struct elem{
        int val;
        ilist_t *elem;
} ilist_t;
ilist_t * append(ilist_t *testa, int num);
int is_prime(num);
ilist_t * solo_primi(int arr[], int dim){
        int i;
        ilist_t *h = NULL;
        for(i=0; i<dim; i++){
                if(is_prime(arr[i])!=0)
                        h = append(h, arr[i]);
}
return h;
}
```

Esercizio 2 (versione 1):

```
/* sottoprogramma che riceve array e retituisce tutti e soli i numeri primi nell'array */
typedef struct elem{
        int val;
```

```
        ilist_t *elem;
} ilist_t;
ilist_t * inserisci_ordinato(ilist_t *testa, int num);
ilist_t * find(ilist_t *testa, int num);
int is_prime(num);
ilist_t * solo_primi(int arr[], int dim){
        int i;
        ilist_t *h = NULL;
        for(i=0; i<dim; i++){
                if(is_prime(arr[i])!=0 && find(h, arr[i])==NULL)
                        h = append(h, arr[i]);
}
return h;
}
```

Esercizio 3:

```
/* programma che prende input valori interi che termina con '0', visualizza in ordine inverso */
#define STOP 0
typedef struct elem{
        int val;
        ilist_t *elem;
} ilist_t;
int main(int argc, char **argv){
        ilist_t *h = NULL;
        int val;
        scanf("%d", &val);
        while(val!=STOP){
                h = push(h, val);
                scanf("%d", &val);
        }
        for(p=h; p!=NULL; p=p->val)
                printf("%d", p->val);
        printf("\n");
        freelist(h);
        return 0;
}
```

Sttoprogrammi utili per le liste:

- 'push()' inserimento in testa alla lista
    1. richiedere memoria con 'malloc' e ricevo l'indirizzo
    2. assegno all'indirizzo il nuovo valore
    3. dico dove deve guardare il nuovo primo elemento (all'ex primo)
    4. sposto la testa e dico che guarda al nuovo primo elemento
    5. se la lista è vuota funziona comunque correttaeìmente

```
ilist_t * push(ilist_t *head, int num){
        ilist_t *n;
        n = (ilist_t*)malloc(sizeof(ilist_t));                /* punto 1 */
        if(n!=NULL){
                n->val = num;                /* punto 2 */
                n->next = head;                /* punto 3 */
                head = n;                        /* punto 4 */
        }else
                printf("push: errore di allocazione\n");
        return head;
}
```

- 'append()' inserimento alla fine
  1. richiedere memoria con 'malloc' e ricevo l'indirizzo
  2. assegnare valore da appendere alla struttura uscita dalla malloc
  3. assegnare puntatore ultimo elemento a null
  4. scorrere lista fino all'ultimo elemento e vedere che punta a 'NULL' (gestire anche caso lista vuote)
  5. assegnare puntatore ex ultimo elemento e risultato malloc
  6. se lista vuota 'append()' si riduce alla 'push()'

```
ilist_t * append(ilist_t *head, int num){
        ilist_t *n, *el;
        n = (ilist_t*)malloc(sizeof(ilist_t));                /* punto 1 */
        if(n!=NULL){
                n->val = num;                /* punto 2 */
                n->next = NULL;                /* punto 3 */
                if(head == NULL)
                        head = n                /* punto 4 (caso lista vuota) */
                else
                        for(el=head; el->next != NULL; el=el->next) /* fin che non si arriva alla fine della lita */
                                ;                /* punto 4 (caso lista non vuota) */
                el->next = n;                /* punto 5 */
        }else
                printf("append: errore di allocazione\n");
        return head;
}
```

- 'list_length()' lunghezza di una lista

```
int list_lenght(ilist_t *head){
        ilist_t *el;
        int len;
        el = head;
        len = 0;
        while(el!=NULL){
                len++;
                el = el->next;
        }
        return len;
}
```

- 'find_val()' ricerca valore, si: indirizzo —— no: NULL

```
int find(ilist_t *head, int num){
        ilist_t *el, *ris;
        el = head;
        ris = NULL;
        while(el!=NULL && ris==NULL){
                el = el->next;
                if(el->val == num)
                        ris = el;
        }
        return ris;
}
```

- 'delete()' riceve testa lista, elemento lista e cancella elemento

1. Se devo eliminare primo elemento:
   (a) caso semplice
2. Se devo eliminare elemento in mezzo:
   (a) trovare elemento pre
   (b) trovare elemento post
   (c) fare 'free()' del puntatore all'elemento

```
ilist_t * delete(ilist_t *head, int num){
        ilist_t *pre, *del;
        del = head;
        while(head!=NULL && (head->val == n)){                /* punto 1 caso semplice */
                del = n;
                h = h->next;                  /* h=del->next */
                free(del);                   /* punto 3 */
        }
        if(h!=NULL){
                pre = h;
                while(pre->next){
                        if(pre->next->val == num){                        /* 'pre->next' punta all'elemento successivo
                                                                             quindi 'pre->next->val' punta
                                                                             succesivo al pre, quindi l'ele
                                del = pre->next;                        /* punto 1 */
                                pre->next = del->next;                /* punto 2:               pre->next=pre->next->next; *
                                free(del);                                        /* punto 3 */
                        }else
                                pre = pre->next;
                }
        }
        return head;
}
```

5.5.7 File

Esiste carattere invisibile che si mette nella defiene 'EOF' (End Of File).
Quando apro un file devo controllare che l'apertura vada a buon fine, sia se devo leggere che scrivere per nopn sovrascrivere file con stesso nome.
Ricordarsi di chiudere i file perchè il sistema operativo riesce a getsore solo un certo numero di file aperti, poi si comporta in maniera casuale.
Noi impareremo a gestire i file composti da una sequenza di caratteri ASCII, quando guardo un file ASCII si vedono i caratteri, quando si opera si lavora con i binari almeno non si perde efficienza nella conversione.

Sintassi base della lettura dei file:

```
FILE* fopen(char nome_file[], char modalita_apertura[]);
/* Restituisce 'NULL' se fallisce l'apertura */
/* Restituisce il riferimento al file  */

fclose(FILE*);
/* 'fclose' su 'NULL' retituisce errore */

int fscanf(FILE* , ______________ );

int fprintf(FILE* , ______________ );

int feof(FILE* ) /* retituisce !0 se ce 'EOF' else 0 */

```

```c
fgets(char[], int , FILE* )

fscanf(stdin, "%d", &val);                /* come scanf() */
fprintf(stdout, "%d\n", val);         /* come printf() */

fread(void* , size_t , int , FILE*)
fwrite(void* , size_t , int , FILE*)
/* r: dove sono i dati        \\ w: dove metto i dati */
/* size_t: dimesione del singolo dato */
/* int: dimensione del singolo dato */
/* FILE*: il riferimento del file */
```

Sintassi generica:

```c
/* read */
fscanf(fin, "%c", &c);
while(!feof(fin)){

        /* istruzioni */

fscanf(fin, "%c", &c);
}

/* write */
ris(fin, "%c", &c);
while(ris==1){

        /* istruzioni */

ris = fscanf(fin, "%c", &c);
}
```

Esercizio con lettura file:

```c
/* programma chiede inserire nome file testo ASCII e conta e visalizza
numero caratteri e carattere piu grande */
#define LMAX 40
int main(int argc, char *argv[]){
        FILE* fin;
        char alpha, cmax;
        int nchar, ris;
        char nome[LMAX+1];
        gets(nome);
        fin = fopen(nome, "r")                /* "r" read, "w" write, "rb" read binario*/
        if(fin){
                ris = fscanf(fin, "%c", &alpha);
                if(ris==1){                /* if(!feof(fin)) */
                        cmax = alpha;
                        nchar = 1;
                        while(fscanf(fin, "%c", &alpha)==1){        /* retiruisce 1 se non 'EOF' */
                                /* lavoro sul dato letto */
                                nchar++;
                                if(alpha>cmax)
                                        cmax = alpha;
                                }
                }
                else{
                        /* file c'e ma è vuoto */
                        nchar = 0;
                }
```

```
                fclose(fin);
                printf("%d\t[%c]\n", nchar, cmax);
        }
        else
                printf("problemi apertura file %s\n", nome);
        return 0;
}
```

Esercizio con lettura file:

```
/* programma chiede inserire nome file testo ASCII e conta e visalizza
numero caratteri e carattere piu grande */
#define LMAX 40
int main(int argc, char *argv[]){
        FILE* fin;
        char alpha, cmax;
        int nchar, ris;
        char nome[LMAX+1];
        gets(nome);
        fin = fopen(nome, "r")                /* "r" read, "w" write, "rb" read binario*/
        if(fin){
                ris = fscanf(fin, "%c", &alpha);
                if(ris==1){                /* if(!feof(fin)) */
                        cmax = alpha;
                        nchar = 1;
                        while(fscanf(fin, "%c", &alpha)==1){        /* retiruisce 1 se non 'EOF' */
                                /* lavoro sul dato letto */
                                nchar++;
                                if(alpha>cmax)
                                        cmax = alpha;
                                }
                }
                else{
                        /* file c'e ma è vuoto */
                        nchar = 0;
                }
                fclose(fin);
                printf("%d\t[%c]\n", nchar, cmax);
        }
        else
                printf("problemi apertura file %s\n", nome);
        return 0;
}
```

Esercizio sui file binario:

```
/* realizzare sottoprogramma che riceve nome file in binario e array di interi
sottoprogramma legge i dati e li mette nell'array e dice quanti dati ci ha messo */
int conta_dati(char nome_file[], int v[]){
        int nval;
        FIL* fin;
        if(fin = fopen(nomef, "rb")){
                fread(&nval, sizeof(int), 1, fin);
                fread(v, sizeof(int), nval, fin)
                fclose(fin);
        }
        else{
                printf("problemi di accesso al file %s\n", nome f);
                nval = 0;
        }
```

```
        return nval;
}
```

```
/* programma che acquisisce valori di lunghezza ignota che termina con valore
inferiore a 1 file salva numeri primi nel file 'primi.csv'*/
/* file.csv è comma separeted value \\ file.ssv è space seoareted value*/

#define SOGLIA 1
#define FNAME "primi.csv"
int main(int argc, char *argv[]){
        FILE* fo;
        int val;
        fo = fopen(FNAME, "w")                /* "r" read, "w" write, "rb" read binario*/
        if(fo){
                scanf("%d", &val);
                while(val>=SOGLIA){
                        if(is_prime(val)){
                                fprintf(fo, "%d\n", val);        /* al posto di '\n' mettere ',' */
                        scanf("%d", &val);
                        }
                }
                fclose(fo);
        }
        else
                printf("problemi apertura file %s\n", FNAME);
        return 0;
}
```

5.5.8 Varibili globali

Per ora abbiamo usato solo programmi con variabili locali, che sono visibili solo nella singola funzione.
Per il passaggio ad altre funzioni si usa o passaggio per valore o per indirizzo.

Definition 5.2 (Variabili globali)**.** *:*
Sono variabili con visibilità potenziata che vengono dichiarate fuori dal main e fuori da qualsiasi sottoprogramma (vengono dichiarate dopo define, typedef e prototipi ma prima del main).

5.6 Algoritmi non immediati

Rotazione in senso orario di una matrice quadrata (indici array multidimensionale):

```
/* inserire dimensione array bidimensionale quadrato di dim_max(M)=10
crea e visulaizza matrice ruotatta in senso orario*/
#include <stdio.h>
#define MAX 10
int main(int argc, char*argv[]){
        int dim, i, j, mat[MAX][MAX], mat_r[MAX][MAX];
        do
                scanf("%d", &dim);
        while (!(dim >= 2 && dim <= 10));
        for(i=0; i<dim; i++){
                for(j=0; j<dim; j++)
                        scanf("%d", &mat[i][j]);
        }
        for(i=0; i<dim; i++){
                for(j=0; j<dim; j++)
```

```
                        /* rotazione array */
                        mat_r[i][j] = mat[dim-j-1][i];
        }
        printf("\n");
        for(i=0; i<dim; i++){
                for(j=0; j<dim; j++)
                        printf("%d ", mat_r[i][j]);
                printf("\n");
        }
        printf("\n");
        return 0;
}
```

Ricerca di elemento di un vettore in un altro vettore (ricerca fra elementi di insieme):

```
/* programma che conta vocali, utente inserisce solo carartteri minuscoli*/
#include<stdio.h>
#include<string.h>/* con errore???? */
#define MAX 100
#define INIZIO 'a'
#define VOC 5
#define FINE 'z'
int main(int argc, char *argv[]){
        char frase[MAX+1], set[VOC+1];
        int i, j, num, trovato;
        gets(frase);
        num = 0;
        for(i=0; frase[i]!='\0'; i++){
                trovato = 0;
                for(j=0; set[j]!='\0' && trovato==0; j++) /* (!trovato) <==> (trovato != 0) */
                        if(frase[i]==set[j])
                                trovato = 1;
                num += trovato;
        }
        printf("vocali: %d\n", num);
        return 0;
}
```

6 Esercitazione

6.1 Esercitazione in autonomia

6.1.1 Esercitazione del 21/10/2022

- Esercizio 1:

```c
/* ## Esercizio 1 - Padding
Si vuole rappresentare a video un valore naturale `num` utilizzando un numero a
scelta di cifre `k` inserendo `0` nelle posizioni più significative, fino a raggiungere
la dimensione desiderata. Per esempio, volendo rappresentare `842` su `5` cifre, si
ottiene `00842`.
Scrivere un programma che acquisisce due valori interi entrambi strettamente positivi
(e finché non è così richiede il valore che non rispetta il vincolo) `num` e `k`, quindi
rappresenta `num` su `k` cifre. Se `k` è minore del
numero di cifre presenti in `num`, il programma visualizza il valore `num` come è. Dopo
il valore visualizzato, mettere un `'a-capo'`. */
#include <stdio.h>
#define BASE 10
#define CAR 0
int main(int argc, char *argv[]){
        int numi, numf, cifre, i;
        do
                scanf("%d", &numi);
        while(numi<=0);
        do
                scanf("%d", &cifre);
        while(cifre<=0);
                numf = numi;
        i = 0;
        while(numf>0){
                i++;
                numf /= BASE;
        }
        cifre -= i;
        for(i=0; i<cifre; i++)
                printf("%d", CAR);
        printf("%d\n", numi);
        return 0;
}
```

- Esercizio 2:

```c
/* ## Esercizio 2 - Super Mario
Nella preistoria dei videogiochi in Super Mario
della Nintendo, Mario deve saltare da una piramide di
blocchi a quella adiacente.
Proviamo a ricreare le stesse piramidi in C, in
testo, utilizzando il carattere cancelletto (`#`)
come blocco, come riportato di seguito. In realtà
il carattere `#` è più alto che largo, quindi le
piramidi saranno un po' più alte.

Notate che lo spazio tra le due piramidi è sempre costituito da __2__ spazi,
indipendentemente dall'altezza delle piramidi. Inoltre, alla fine delle piramidi __non
ci devono essere spazi__.
L'utente inserisce l'altezza delle piramidi, che deve essere un valore strettamente
positivo e non superiore a 16. In caso l'utente inserisca un valore che non rispetta
questi vincoli, la richiesta viene ripetuta.*/
```

```c
#include <stdio.h>
#define MAX 16
#define MIN 1
#define ARIA_CENTRO 2
#define BLOCCHI '#'
#define ARIA ' '
int main (int argc, char *argv[]){
    int piani, i, j, nAria, nBlocchi;
    do scanf("%d", &piani);
    while (piani < MIN || piani > MAX);
    for (i = 1; i <= piani; i++){
        nAria = piani - i;
        nBlocchi = piani - nAria;
        for (j = 0; j < nAria; j++)
            printf("%c", ARIA);
        for (j = 0; j < nBlocchi; j++)
            printf("%c", BLOCCHI);
        for (j = 0; j < ARIA_CENTRO; j++)
            printf("%c", ARIA);
        for (j = 0; j < nBlocchi; j++)
            printf("%c", BLOCCHI);
        printf("\n");
    }
    return 0;
}
```

- Esercizio 3:

```c
/* ## Esercizio 3 - Troncabile primo a destra
Scrivere un programma che acquisisce un valore intero strettamente positivo, e finché
non è tale lo richiede. Il programma analizza il valore intero e visualizza 1 nel caso
sia un `troncabile primo a destra`, 0 altrimenti.
Un numero si dice troncabile primo a destra se il numero stesso e tutti i numeri che si
ottengono eliminando una alla volta la cifra meno significativa del numero analizzato al
passo precedente, sono numeri primi.
Per esempio, se il numero iniziale è 719, i numeri che si ottengono \eliminando una
alla volta la cifra meno significativa del numero analizzato al passo precedente .."
sono 71 e 7.
Dopo il valore visualizzato, mettere un `'a-capo'`. */
#include <stdio.h>
#define BASE 10
int primo(int val){ /* '1' primo - '0' non primo */
        int i, ris, meta;
        if(val>0){                /* controllo positivita numero */
                ris = 1;
                if(val%2 == 0){
                        ris = 0;
                }
                else{
                        meta = val/2;                /* tanto 20 non è divisibile per 'i>10' sicuro */
                        for(i=3; i<meta && ris != 0; i += 2){
                                if(val%i == 0)
                                        ris = 0;
                        }
                }
        }
        else
                ris = -1;
        return ris;
}
```

```c
int main(int argc, char* argv[]){
        int num, tpd;
        do
                scanf("%d", &num);
        while(num<=0);
        tpd = 1;
        num /= BASE;
        while(num>0 && tpd==1){
                if(primo(num) != 1)
                        tpd = 0;
                num /= BASE;
        }
        printf("%d\n", tpd);
        return 0;
}
```

- Esercizio 4:

```c
/* ## Esercizio 4 -- Tartaglia
Scrivere un programma che mostra a video il triangolo di Tartaglia di dimensione
chiesta all'utente (massimo 10). Il programma deve eseguire un controllo di validità
sulla dimensione chiesta all'utente e nel caso di valore non valido richiederla.
Esempio: il triangolo di Tartaglia di dimensione 5 (valore inserito dall'utente) è:
```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
``` */
#include<stdio.h>
#define MAX 10
int main(int argc, char *argv[]){
        int i, j, dim, trng[MAX][MAX];
        do
                scanf("%d", &dim);
        while(dim<=0);
        for(i=0; i<dim; i++){
                for(j=0; j<=i; j++){
                        if(j==0 || j==i)
                                trng[i][j]=1;
                        else
                                trng[i][j]=trng[i-1][j-1]+trng[i-1][j];
                }
        }
        for(i=0; i<dim; i++){
                for(j=0; j<=i; j++){
                                printf("%d ", trng[i][j]);
                }
                printf("\n");
        }
        return 0;
}
```

- Esercizio 5:

```c
/* ## Esercizio 5 -- Rotazione stringa
Scrivere un programma che acquisisce una stringa `seq1` di massimo 50 caratteri e un
```

```
numero intero `n`. Il valore `n` deve essere minore della lunghezza della stringa e
fino a quando non è tale lo richiede all'utente. In seguito il programma crea una nuova
stringa `seq2` che contiene la rotazione verso destra di `seq1` di `n` posizioni e la
visualizza. La nuova stringa va creata, non è sufficiente visualizzare il risultato.
Esempio:
ingresso:
alfabeto 2
seq2 e uscita: toalfabe */
#include <stdio.h>
#include <string.h>
#define LEN 50
int main (int argc, char *argv[]){
    char frase[LEN + 1], reverse[LEN + 1];
    int i, len, ruota, j;
    gets(frase);
    scanf("%d", &ruota);
    len = strlen(frase);
    for (i = 0; i < len; i++){
                if(j<0)
                        j = i - ruota + len;
        else
                        j = i - ruota;
        reverse[i] = frase [j];
    }
    reverse[len] = '\0';
    printf("%s\n", reverse);
    return 0;
}
```

- Esercizio 6:

```
/* ## Esercizio articolato **** Complessità superiore alla media degli altri esercizi
Il genio della lampada vi fa recapitare un messaggio in cui vi rivela l'andamento
in borsa dei titoli della società GEN-IA-LE per i prossimi due anni, e questa è
l'occasione per guadagnare qualche soldo. Il messaggio purtroppo si smaterializzerà
nell'arco di un'ora per cui è necessario riuscire a trovare velocemente come utilizzare
l'informazione.
L'informazione consiste nel fornire il prezzo del titolo iniziale, e dire, per
ogni giorno, la variazione del valore del titolo alla chiusura rispetto al giorno
precedente. Pensando ad una finestra di soli 11 giorni, per semplicità, l'informazione a
disposizione è la seguente:
18
+4 -6 +3 +1 +3 -2 +3 -4 +1 -9 +6 +2 +1 -5 +3 +1 +2 0 -1 -1
Si consideri di acquistare la mattina all'apertura, alla quotazione di chiusura del
giorno prima, e di vendere a termine giornata, quindi alla quotazione di chiusura del
giorno.
In ingresso il programma riceve un primo intero che rappresenta il valore iniziale
del titolo, quindi acquisisce 20 valori interi che costituiscono le variazioni di
quotazione del titolo dei 20 giorni successivi.
Realizzate il programma che vi consente di guadagnare di più, individuando il giorno in
cui acquistare (`a`) e quello in cui vendere (`v`).
Per semplicità, realizzate prima un programma che ipotizzi che il vostro capitale sia
infinito, e realizzate poi un programma che riceva in ingresso anche il capitale `C` a
vostra disposizione (per cui il numero di azioni che potete acquistare dipende dal loro
valore). */

#include <stdio.h>
#define GIORNI 20
#define OPZ_1 "perdita"
#define OPZ_2 "capitale insufficente"
```

```c

int main (int argc, char *argv[]){
    int cambi[GIORNI], iTemp, guadagno, guadagnoTemp, capI, preI, iStart, iFinish, i, found;

    scanf("%d", &preI);
    for (i = 0; i < GIORNI; i++)
        scanf("%d", &cambi[i]);
    scanf("%d", &capI);

    found = 0;
    for (i = 0; i < GIORNI && !found; i++){
        if (preI <= capI){
            iStart = i;
            iFinish = i;
            guadagno = cambi[i];
            guadagnoTemp = cambi[i];
            iTemp = i;
            found = 1;
        }
        preI += cambi[i];
    }

    for (; i < GIORNI; i++){
        guadagnoTemp += cambi[i];
        if (cambi[i] > guadagnoTemp && preI <= capI){
            guadagnoTemp = cambi[i];
            iTemp = i;
        }
        if (guadagnoTemp > guadagno){
            guadagno = guadagnoTemp;
            iStart = iTemp;
            iFinish = i;
        }
        preI += cambi[i];
    }

    if (found)
        if (guadagno > 0)
            printf("%d %d", iStart, iFinish);
        else
            printf(OPZ_1);

    else
            printf(OPZ_2);
    printf("\n");
    return 0;
}
```

6.2 Esercizi del laboratorio

6.2.1 Esercitazione 27/09/2022

- Esercizio 1:

```c
/* programma + - x / calcolaterice polacca in caso di operatore invalido retituisca x */
#include<stdio.h>
#define INV 999999
int main(int argc,char*argv[]){
        int a,b;
        float ris;
        char operatore;
        scanf("%c\n",&operatore);
        switch(operatore){
                case '+':
                        ris=a+b;
                case '-':
                        ris=a-b;
                case '*':
                        ris=a*b;
                case '/':
                        ris=a/b;
                default:
                        ris=INV;
        }
        printf("%f",ris);
        return 0;
}
```

- Esercizio 2:

```c
/* data di nascita e data attuale gg mm aa ==> calcola la data in anni */
#define M_A 12
#define G_A 365
#include<stdio.h>
int main(int argc, char*argv[]){
        int ggp, mmp, aap, gga, mma, aaa, ris;
        float p1,p2;
        scanf("%d\t %d\t %d\n        %d\t %d\t %d\n", &ggp, &mmp, &aap, &gga, &mma, &aaa);
        p1=ggp/G_A+mmp/M_A+aap;
        p2=gga/G_A+mma/M_A+aaa;
        ris=p2-p1;
        printf("%d\n",ris);
        return 0;
}
```

- Esercizio 3:

```c
/* leggere 3 numeri interi, determinare sequenza numeri monotona non decrescente */
#include<stdio.h>
int main(int argc,char*argv[]){
        int a, b, c, var;
        scanf("%d %d %d", &a, &b, &c);
        if(a<b){
                var=a;
                a=b;
                b=var;
        }
```

```
        if(a<c){
                var=a;
                a=c;
                c=var;
        }
        if(b<c){
                var=b;
                b=c;
                c=var;
        }
        printf("%d %d %d\n",c,b,a);
        return 0;
}
```

- Esercizio 4:

```
/*  acquisito numero intero visualizzi '1' se pari e '0' se dispari*/
#include <stdio.h>
#define PARI 0
#define DISPARI 1
int main(int argc, char*argv[]){
        int num, ris;
        scanf("%d",&num);
        if(num%2==0) /* Resto della divisione */
                ris=PARI;
        else
                ris=DISPARI;
        printf("%d\n",ris);
        return 0;
}
```

- Esercizio 5:

```
/* prezzo costo e vendita, verificare profitto perdita; restituire margine in valore assoluto restituisce '+','-','x' */
#include<stdio.h>
#define POS '+'
#define NEG '-'
#define PAR 'x'

int main(int argc,char*argv[]){
        float costo, vendita, profitto;
        char out;
        scanf("%f\n%f",&costo,&vendita);
        profitto=vendita-costo;
        if(out>0)
                out=POS;
        else{
                if(out<0)
                        out=NEG;
                else
                        out=PAR;
        }
        if(profitto<0)
                profitto=-profitto;
        printf("%c\n%f\n",out,profitto);
        return 0;
}
```

- Esercizio 6:

```c
/* corso con 2 esami, entrambi positivi e media valori arrotondata per eccesso (int 2 valori interi [0-30])
- 1 promosso - 0 bocciato - media voto*/

#include<stdio.h>
#define PROM 1
#define BOCC 0
#define NUM_ESAMI 2
int main(int argc,char*argv[]){
        int voto1, voto2, avg;
        char ris;
        scanf("%d\n%d",&voto1,&voto2);
        if (voto1>=18 && voto2>=18){
                ris=PROM;
                avg=(voto1+voto2+1)/NUM_ESAMI;
                printf("%d\n",avg);
        }
        else
                ris=BOCC;
        printf("%c",ris);
        return 0;
}
```

- Esercizio 7:

```c
/* progamma chiede a b float e calcola e visualizza soluzione, se impossibile programma y se indeterminata z */
#include <stdio.h>
#define IMP 'y'
#define IND 'z'
int main(int argc,char*argv[]){
        float a, b, sol;
        char ris="";
        scanf("%f\n%f",&a,&b);
        if(a==0 && b==0){
                ris=IND;
                printf("%c",ris);
        }
        else if(a==0 && b!=0){
                ris=IMP;
                printf("%c",ris);
        }
        else{
                sol=-b/a;
                printf("%f\n",sol);
        }
        return 0;
}
```

6.2.2 Esercitazione 03/10/2022

- Esercizio 1:

```c
/* acquisito numero visualizza la somma delle sue cifre */
#include <stdio.h>
#define BASE 10
int main(int argc,char*argv[]){
        int num, sum;
```

```
        scanf("%d",&num);
        if(num<0)
                num = -num;
        sum = 0;
        while(num!=0){
                sum += num%BASE;
                num /= BASE;
        }
        printf("%d\n",sum);
        return 0;
}
```

- Esercizio 2:

```
/* acquisire numero positivo e fin che tale lo richiede, determini parete inter a della radice quadrata */
#include <stdio.h>
int main(int argc,char*argv[]){
        int num, rad, test;
        do
                scanf("%d",&num);
        while(num<0);
        rad = 0;
        test = 0;
        while(test<=num){
                rad++;
                test = rad*rad;
        }
        rad -= 1;
        printf("%d\n",rad);
        return 0;
}
```

- Esercizio 3:

```
/* num1 positivo saldo conto corrente e num2 tasso interesse
banca num3 saldo desiderato e determini dopo quanti anni*/
#include <stdio.h>
int main(int argc,char*argv[]){
        float saldo_i, tasso, saldo_f;
        int year;
        do
                scanf("%f",&saldo_i);
        while(saldo_i<=0);
        do
                scanf("%f",&tasso);
        while(tasso<=0);
        do
                scanf("%f",&saldo_f);
        while(saldo_f<saldo_i);
        year = 0;
        while(saldo_i<saldo_f){
                saldo_i += saldo_i*tasso;
                year++;
        }
        printf("%d\n",year);
        return 0;
}
```

- Esercizio 4:

```c
/* numero palindoromo quando numero è uguale sia letto da destra che sinistra
(se inizia con zero non è palindromo), se palindormo '1' se non palindromo '0'*/
#include <stdio.h>
#define PAL 's'
#define NO_PAL 'n'
#define BASE 10
int main(int argc,char*argv[]){
        int num, ris, supp, cifra, inverso;
        do{
                scanf("%d",&num);
        }while(num<=0);
        supp = num;
        cifra = supp%BASE;
        inverso = 0;
        if(cifra==0)
                ris = NO_PAL;
        else{
                while(supp>0){
                        inverso=inverso*BASE+cifra;
                        supp /= BASE;
                }
                if(inverso==num)
                        ris = PAL;
        else
                ris = NO_PAL;
        }
        printf("%c\n",ris);
        return 0;
}
```

- Esercizio 5:

```c
/* se num è doppio del precedente stamparli enrtrambi else vai avanti, esci dal ciclo se metti zero*/
#include <stdio.h>
#define STOP 0
int main(int argc,char*argv[]){
        int prec, succ;
        scanf("%d",&prec);
        while(prec!=STOP){
                scanf("%d",&succ);
                if(2*prec == succ)
                        printf("%d\t%d\n",prec,succ);
                prec = succ;
        }
        return 0;
}
```

- Esercizio 6 (versione 1):

```c
/* calcola e MCD MCM */
#include <stdio.h>
int main(int argc,char*argv[]){
        int a,b,mcm,mcd;
        do
                scanf("%d",&a);
        while(a<=0);
```

```
        do
                scanf("%d",&b);
        while(b<=0);
        if(a>b){
                mcd = a;
                mcd = b;
        }
        else{
                mcd = b;
                mcd = a;
        }
        while(a%mcd==0 && b%mcd==0)
                mcd--;
        while(mcm%a==0 || mcm%b==0)
                mcm++;
        printf("%d %d",mcd,mcm);
        return 0;
}
```

- Esercizio 6 (versione 2 con algoritmo di euclide):
  L'algoritmo di Euclide si basa sul seguente teorema: Dati due numeri naturali a e b, entrambi maggiori di 1 con $a > b$: se b è un divisore esatto di a, b è ovviamente il massimo comun divisore tra i due numeri; altrimenti, detto r il resto della divisione tra a e b, il MCD tra a e b è uguale al MCD tra b e r:

```
/* MCD e MCM con algortmo di euclide */
#include <stdio.h>
int main (int argc, char*argv[]){
        int num1, num2, tmp1, tmp2, resto, mcd, mcm;
        do
                scanf("%d",&num1);
        while (num1<=0);
        do
                scanf("%d",&num2);
        while (num2<=0);
        if (num1 > num2){
                tmp1 = num1;
                tmp2 = num2;
        } else {
                tmp1 = num2;
                tmp2 = num1;
        }
        resto = tmp1 % tmp2;
        while (resto != 0){
                tmp1 = tmp2;
                tmp2 = resto;
                resto = tmp1 % tmp2;
        }
        mcd = tmp2;
        mcm = num1 * num2 / mcd;
        printf("%d\t%d\n", mcd, mcm);
        return 0;
}
```

6.2.3 Esercitazione 13/10/2022

- Esercizio 1:

```c
#include<stdio.h>
#define MAX 100

int main(int argc, char *argv[]){
        int valori[MAX][2],n_val,i, j, max;
        do
                scanf("%d",&n_val);
        while(!(n_val < MAX));
        i = 0;
        do{
                scanf("%d",&valori[i][0]);
                i++;
        }while(i < n_val);
//calcola valore piu frequente
        for(i=0; i<n_val; i++)
                valori[i][1] = 1;
        j = i;
        max = 1;
        do{
                if(valori[i][0]==valori[j+1][0])
                        valori[j][1] += valori[i][1];
                j++;
                if(max>valori[j][0])
                        max = valori[j][0];
        }while(!(valori[i][0]==valori[j+1][0] || j>n_val-1));

        //lunghezza sequenza senza frequente
        for(i=0; i<n_val; i++, j++){
                if(valori[i][0]==max)
                        j--;
        }
        j++;
        printf("%d\n", j);
        //sequenza valori senza piu frequente
        for(i=0; i<n_val; i++, j++){
                while(valori[i][0]==max)
                        i++;
                printf("%d\t",valori[i][0]);
        }
        printf("\n");
        return 0;
}
```

- Esercizio 2: Si scriva un programma in linguaggio C (ANSI 89) che acquisito un numero maggiore o uguale a zero (e fino a che non è tale lo richiede), ne calcoli e visualizzi il fattoriale. NOTA: si ricordi che fattoriale di 0 è 1.

```c
/* numero >= 0 visualizzi fattoriale !0=1 */
#include<stdio.h>
int main(int argc, char*argv[]){
        int num, fat, i;
        do
                scanf("%d",&num);
        while(num<0);
        fat = 1;
                for(i=num; i>0; i--)
                        fat *= i;
        }
        printf("%d\n",fat);
```

```
        return 0;
}
```

- Esercizio 3: Scrivere un programma in linguaggio C (ANSI 89) che chiede all'utente una sequenza di 10 numeri interi. Il programma controlla che non vi siano duplicati all'interno della sequenza data e visualizza l'esito del test, cioè 1 se non ci sono duplicati, 0 altrimenti.

```c
/* 10 numeri interi e vede se ci sono duplicati - '1' non ci sono doppioni - '0' no */
#include<stdio.h>
#define DIM 10
#define SI 0
#define NO 1
int main(int argc, char*argv[]){
        int valori[DIM], i, j, ris;
        for(i=0; i<DIM; i++)
                scanf("%d",&valori[i]);
        ris = SI;
        for(j=0; j<DIM && ris==SI; j++){
                for(i=0; i<DIM && ris==SI; i++){
                        if(valori[i]==valori[j]);
                                ris = NO;
                }
        }
        printf("%d\n",ris);
        return 0;
}
```

- Esercizio 4: Si scriva un programma in linguaggio C (ANSI 89) che acquisisce il prezzo di massimo 50 prodotti, espresso mediante un valore reale, che termina quando l'utente inserisce un valore non positivo (nel caso di 50 prodotti, verrebbe fornito come 51 prezzo). Il programma acquisisce quindi la quantità di denaro disponibile. Il programma calcola e visualizza il numero massimo di prodotti acquistabili (si vuole massimizzare la quantità di prodotti).

```c
/* inserire al massimo 50 prezzi fino a 50 prodotti (fino  a numero non positivo)
il 51esimo elemento è la spesa massima
*/
#include <stdio.h>
#define MAX 50
int main(int argc, char*argv[]){
        int prezzi[MAX], i , j,  k, num_pezzi, pezzi;
        float  temp, max_spesa, sum_spesa;
        i = 0;
        scanf("%f",&temp);
        while(!(i<MAX && temp>=0.0)){
                temp = prezzi[i];
                scanf("%f",&temp);
                i++;
        }
        pezzi = i--;
        while(max_spesa<0.0);
        for(j=0; j<pezzi-1; j++){/* ordinamento bubble sort */
                for(k=0; k<pezzi-1-j; k++){
                        if(prezzi[k]>prezzi[k+1]){/* ripetizione accorciata*/
                                temp = prezzi[k];
                                prezzi[k] = prezzi[k+1];
                                prezzi[k+1] = temp;
                        }
                }
```

```
        }
        i = 0;
        num_pezzi = 0;
        sum_spesa = 0.0;
        while(i<pezzi && sum_spesa<=max_spesa){
                if(sum_spesa+prezzi[i]<=max_spesa){
                        sum_spesa += prezzi[i];
                        num_pezzi++;
                }
        }
        printf("%d\n",num_pezzi);
        return 0;
}
```

- Esercizio 5: Si scriva un programma in linguaggio C (ANSI 89) che acquisisce un array bidimensionale quadrato di dimensione 4. Terminata l'acquisizione, il programma calcola e visualizza 1, se i valori sulla diagonale sono strettamente crescenti, 0 altrimenti.

```
/* digonale matrice quadrata strettamente crescenti - '1' si '0' no */
#include <stdio.h>
#define SIZE 4
int main(int argc; char*argv[]){
        int mat[SIZE][SIZE], r, c, ris, last;
        ris = 1;
        for(r=0; r<SIZE; r++){
                for(c=0; c<SIZE; c++){
                        scanf(%d",mat[r][c]);
                }
        }
        last = mat[0][0];
        for(i=0; i<SIZE; i++){
                if(mat[i][i]<=last)
                        ris = 0;
                last = mat[i][i];
        }
        printf("%d",ris);
        return 0;
}
```

- Esercizio 6: Scrivere un programma in linguaggio C (ANSI 89) che chiede all'utente di inserire i dati interi di un array bi-dimensionale quadrato di dimensione 3 e visualizza 1 se la matrice è simmetrica, 0 altrimenti.

```
/* controllare controllare se matrice 3x3 è simmetrice M=M^T - '1' simmetrica - '0' no*/
#include<stdio.h>
#define DIM 3
int main(int argc, char*argv[]){
        int i, j, mat[DIM][DIM];
        for(i=0; i<DIM; i++)
                for(j=0; j<DIM; j--)
                        scanf("%d",mat[i][j]);

        for(i=0; i<DIM; i++)
                for(j=0; j<DIM; j--){
                        if(mat[i][j]!=mat[j][i])
                        ris = 0;
                }
        printf("%d\n",ris);
```

```
        return 0;
}
```

- Esercizio 7: Un algoritmo molto basilare di compressione di una stringa consiste nel sostituire ogni gruppo di caratteri identici consecutivi con il carattere seguito dal numero delle sue occorrenze. Si scriva un programma in linguaggio C (ANSI 89) che acquisita in ingresso una stringa di al massimo 50 caratteri la comprima. Si consideri che nella stringa compressa il conteggio delle apparizioni di un carattere possa essere al massimo 9. Al termine dell'elaborazione si vuole avere a disposizione sia la stringa originale, sia quella compressa che viene poi visualizzata. Si assuma che la stringa sia composta solo da caratteri minuscoli dell'alfabeto inglese.

```c
/* compressione stringa da 'aaabbbbbccaaa' ==> 'a3b4c3a3'*/
#include <stdio.h>
#include <string.h>
#define INIZIO a
#define FINE z
#define MAX_LEN 50
#define MAX_REPS 9
#define MAX_LEN_ZIP (MAX_LEN*2)
#define ZERO_CHAR '0'
int main(int argc, char*argv[]){
        int i, j, ;
        char testo[MAX_LEN+1], compr[MAX_LEN_ZIP+1], ourChar;
        scanf("%s", testo);
        ourChar = testo[0];
        cont = 1;
        lenZip = 0;
        for(i=1; i!=?\0'; i++){
                if(ourChar!=testo[i] || cont=MAX_REPS){
                        compr[lenZip] = ourChar;
                        compr[lenZin+1] = cont+ZERO_CHAR;
                        lenZip += 2;
                        ourChar = testo[i];
                        cont = 1;
                }
                else
                        cont++;
        }
        compr[lenZip] = ourChar;
        compr[lenZin+1] = cont+ZERO_CHAR;
        compr[lenZip+2] = '\0';
        printf("%s\n",compr);
        return 0;
}
```

- Esercizio 8: Scrivere un programma in linguaggio C (ANSI 89) che chiede all'utente una sequenza di al massimo 30 caratteri. Il programma identifica nella sequenza tutte le sotto-sequenze di sole cifre in posizioni consecutive, e visualizza le lunghezze della sotto-sequenza più lunga e di quella più corta. Nel caso la stringa non contenga alcuna cifra, il programma visualizza il messaggio "0 0". ESEMPIO: se per esempio la sequenza di ingresso è "a1245b645c7de45", il programma visualizza i valori 4 e 1 (avendo individuato le sotto sequenze "1245" e "7").

```c
/* visualizzare sequanze cifra piu corta e piu lunga */
#include <stdio.h>
#define MAX 30
int main(int argc, char*argv[]){
        char seq[MAX+1];
        int max, min, i, dim, count;
        scanf("%s", seq);
```

```
        i = 0;
        while(!(seq[i]>='0' && seq[i]<='9') && seq[i]!='\0')
                i++;
        if(seq[i]!='0'){
                dim = i+1;
                while(seq[dim]>=0 && seq[dim]<=9 && seq[dim]!='\0')
                        dim++
                min = dim-1;
                max = dim-1;
                count = 0;
                for(i=dim; seq[i]!='\0'; i++)
                        if(seq[i]>='0' && seq[i]<='9')
                                count++;
                        else if(count){
                                if(count>max)
                                        max = count;
                                else if(count<min)
                                        min= count;
                        count = 0;
                        }
        }
        if(count){
                if(count>max)
                        max = count;
                else if(count<min)
                        min = count;
        }else{
                max = 0;
                min = 0;
        }
        printf("%d%d",min,max);

                                                /* da finire */
        }

        return 0;
}
```

6.2.4 Esercitazione 20/10/2022

- Esercizio 1:

```
/* array bidimensionale in input, trasmette indice di riga  e colonna con somme di valori minimo (MAT 3x3)*/
void mat_irc(mat[][N_COLS}, int nr, int nc, int *nc, int *nr){
        int j, min_r, min_c=0;
        for(j=0; j<nc; j++){
                min_r += mat[0][j]
        }
        index_r = 0;
        for(i = 1; i<nc; i++){
                sum_r = 0;
                for(j=0; j<nc; j++){
                        sum_r += mat[i][j];
                        if(sum_r = min_r){
                                min_r = sum_r;
                                index_r = 1;
                        }
                }
                *ir = index_r;
                min_c = 0;
```

```
                for(j=0; j<nc; j++)
                        min_c += mat[j][0];
                index_c=0;
                for(i=0; i<nc; i++){
                        sum_c += mat[i][j];
                        if(sum_c <= min_c){
                                min_c = sum_c;
                                index_c = j;
                        }
                }
        }
        *ic = index_c;
}
```

- Esercizio 2:

```
* scrivere un programma che ricevuta stringa con password restituisce 1 se
almeno 1 cifra \\ almeno un carattere non cifra e non alfabeto \\ almeno 8 caratteri \\ no 2 uguale consecutivi */
#include<stdio.h>
#define LMAX 100
#define LMIN 8
#define NUMMIN '0'
#define NUMMAX '9'
#define LOINT 'a'
#define LOEND 'z'
#define UPINT 'a'
#define UPEND 'z'

int valida (char pwd[]){
        int i, ris, cifra, alfa, doppia;
        cifra = 0;
        alfa = 0;
        doppia = 0;
        for(i=0; pwd[i]!='\0'; i++){
                if(pwd[i]>=NUMMIN && pwd[i]<=NUMMAX)
                        cifra = 1;
                else if(!(pwd[i]>= LOINT && pwd[i]<=LOEND || pwd[i]>= UPINT && pwd[i]<=UPEND))
                        alfa = 1;
                if(pwd[i]<=pwd[i+1]
                        doppia = 1;
        }
        if(i>=LMIN && alfa && doppia)
                ris = 1;
        else
                ris = 0;
}
int main(int argc, char *argv[]){
        char pwd[LMA+1];
        scanf("%s", pwd);
        printf("%d", valida(pwd));
        return 0;
}
```

- Esercizio 3:

```
/* array utente non sa max - termina con zero, programma restituisce solo valori validi (max 10) */
#include<stdio.h>
#define MAX 100
```

```c
void acquisisci_valori(int num[], int dim_max, int *validi){
        float valore;
        int valoriValidi;
        valoriValidi = 0;
        while(valore!=0){
                if(valoriValidi<dim_max){
                        num[valoriValidi] = valore;
                        valoriValidi++;
                }
                scanf("%f", &valore);
                *validi = valoriValidi;
}
int main(int argc, char *argv[]){
        float numeri[MAX], temp;
        int val_validi;
        acquisisci_valori(numeri, MAX, &val_validi);
        for(i=0; i<val_validi; i++)
                printf("%f", numeri[i]);
        return 0;
}
```

- Esercizio 4:

```c
/* modifica stringa facedno diventare prima lettera maiuscola e altre minuscole */
#include <stdio.h>
#define OFFSET 'A'-'a'
#define L_MAX 100
#define FROM_LOW 'a'
#define TO_LOW 'z'
#define SEP ' '
void modifiche(){
        int i;
        for(i=0; frase[i]=!'\0'; i++){
                if(frase[i]>=FROM_LOW && frase[i]>=TO_LOW;
                                frase[i] += OFFSET;
                }
        }
        while(frase[i]=!'\0'){
                if(frase[i]>=FROM_LOW && frase[i]>=TO_LOW)
                        frase[i] += OFFSET;
                i++;
                while(frase[i]!=SEP && frase[i]!='\0'){


                }
        }


}
int main(int argc, char*argv[]){
        char seq[L_MAX+1];
        gets(seq);
        modifiche(seq);
        printf("%s", seq);
        return 0;
}
```

- Esercizio 5:

```c
/* array di numeri reale con caselle somma di tutte le precedenti */
#include <stdio.h>
#define DIM 10
void somma_precedenti(float num[], float sum[], int val){
        int i, j, risulato;
        if(val>0)
                sum[0]=num[0];
        for(i=0; i<val; i++){
                sum[i] = num[i]+sum[i-1]
        }
}
int main(int argc, char *argv[]){
        float numeri[DIM], somme[DIM], temp;
        int i, validi;
        validi = 0;
        scanf("%d", &temp);
        while(temp>=0 && i<10){
                numeri[validi] = temp;
                scanf("%d", &temp]);
                validi++;
        }
        somma_precedenti(numeri, somme, validi);


        for(i=0; i<len; i++)
                printf("%d", somme[i]);
        return 0;
}
```

- Esercizio 6:

```c
/* sis scirva un sottoprogramma che ricevutoin ingtrersso 10 val interi e coppia di numeri validi se indici
num[i]==num[j] con i<j*/
#include <stdio.h>
#define DIM 10
int n_coppie(int vettore[], int len){
        int i, j, ris, coppie;
        for(i=0; i<len-1; i++){
                for(j=i+1; j<len && ris!=1; j++)
                        ris = 0;
                        if(vettore[i]==vettor[j])
                                ris=1;
                                coppie += ris;
        }
        return coppie;
}
int main(int argc, char *argv[]){
        int numeri[DIM];
        int i, coppie;
        for(i=0; i<NUMS; i++)
                scanf("%d", &numeri[i]);
        coppie = n_coppie(numeri, DIM)
        printf("%d", coppie);
        return 0;
}
```

References