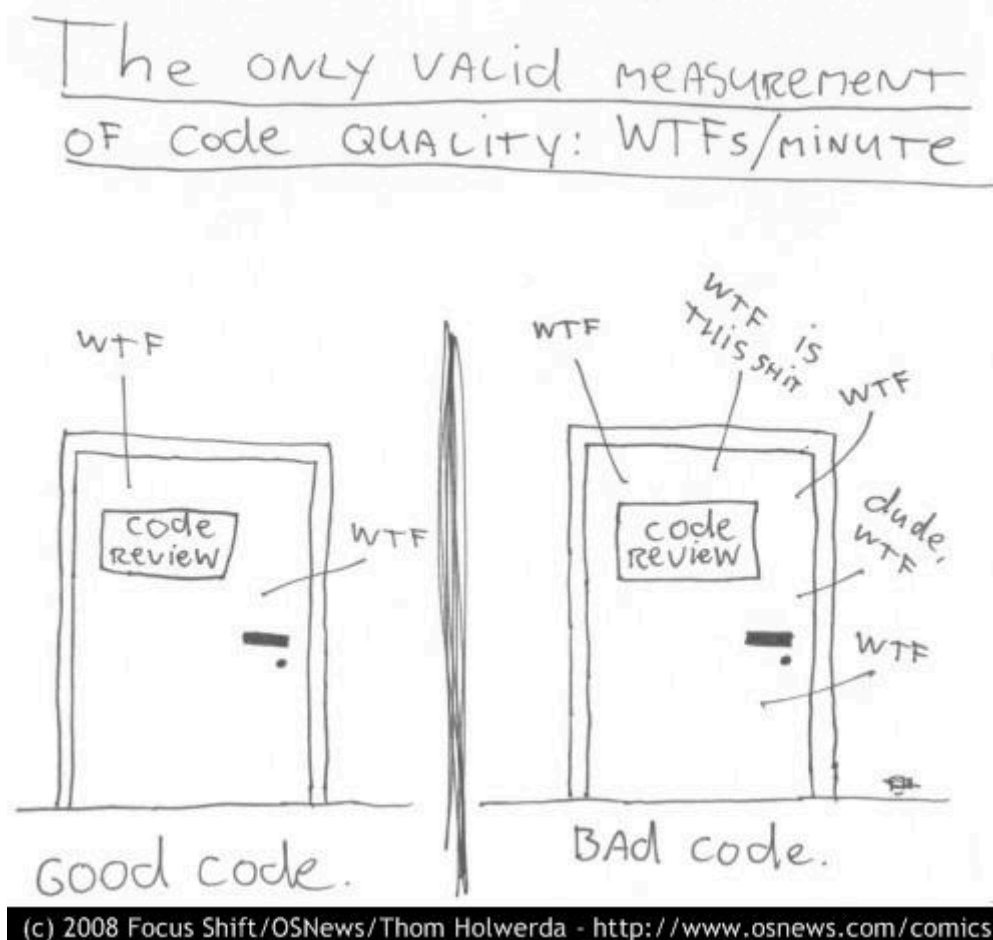


Appunti di Informatica

Classe 3^a



Maria Grazia Maffucci
dal 2021 ad oggi



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/4.0/>.

Contact mariagrazia@maffucci.cc

I am not the owner of these ideas, I only made a pedagogical transposition.

Teaching is care.

"Quality is the result of a million selfless acts of care—not just of any great method that descends from the heavens. That these acts are simple doesn't mean that they are simplistic, and it hardly means that they are easy. They are nonetheless the fabric of greatness and, more so, of beauty, in any human endeavour. To ignore them is not yet to be fully human."

Clean Code - A Handbook of Agile Software Craftsmanship, by Robert C. Martin

Indice

Indice.....	3
Problemi, programmi e algoritmi.....	7
La programmazione strutturata.....	10
Linguaggi di programmazione strutturati.....	11
Le strutture di controllo.....	12
Sequenza.....	12
Selezione.....	12
Iterazione.....	13
Documentazione di progetto iniziale.....	14
Sequenza, selezione, iterazione.....	16
C Tutorial.....	16
Esercizi.....	18
Programmazione modulare.....	33
Funzioni.....	33
C Tutorial.....	33
Passaggio dei parametri per valore.....	34
Somma di due numeri.....	34
Scambio di due numeri.....	37
Passaggio dei parametri per indirizzo.....	40
Scambio di due numeri.....	40
File header definiti dal programmatore.....	43
Header guard.....	43
#ifndef.....	43
#ifdef.....	44
#pragma once.....	45
C Tutorial.....	46
Approfondimenti.....	47
C Tutorial.....	47
Variadic functions: funzioni con un numero variabile di parametri.....	47
Direttive condizionali del pre processore.....	47
Dalla programmazione strutturata alla programmazione modulare.....	48
Decomposizione funzionale - Functional decomposition.....	49
Organizzazione architetturale di un'applicazione.....	50
Encapsulation principle - Principio di incapsulamento.....	51
Esercizi.....	52
Compilazione e interpretazione dei programmi.....	59
C Compiler.....	59
Preprocessing.....	61
Compiling.....	62
Assembling.....	62
Linking.....	63
Riepilogo dei comandi per la compilazione a linea di comando.....	64

Compiler vs Interpreter vs Hybrid compiling.....	64
La struttura della memoria di un processo.....	65
La memoria virtuale di un processo.....	66
The Call Stack.....	70
Learn more about Stack and Heap.....	70
Strutture dati.....	71
Array e stringhe.....	71
C Tutorial - Array.....	71
C Tutorial - String.....	71
Esempio sull'uso di array e stringhe.....	72
Array numerici con singola stringa (es.122 e es.124).....	72
Array paralleli stringhe con elenco di singoli numeri.....	75
Array numerici e di stringhe.....	80
Uso di fgets().....	81
Esercizi.....	82
Struct.....	90
C Tutorial.....	90
Struct e programmazione modulare.....	91
Strutture esplicite.....	91
Esempio di programmazione modulare con strutture esplicite - Persona.....	91
Variabili static.....	94
C Tutorial.....	94
Esempio con strutture esplicite e modificatore static - Array di persone.....	94
Esercizi.....	101
La memoria dinamica.....	107
Array numerici dinamici.....	107
C Tutorial.....	107
Esempio di gestione di un array dinamico di numeri.....	108
Esercizi.....	112
Stringhe dinamiche e array di stringhe.....	113
C Tutorial.....	113
Esempio di array dinamico di stringhe.....	114
Esercizi.....	118
Allocazione dinamica di struct.....	119
C Tutorial.....	119
Esempio di allocazione dinamica di una struct.....	120
Struct e programmazione modulare.....	122
Strutture nascoste.....	122
C Tutorial.....	123
Esempio di programmazione modulare con strutture nascoste.....	124
Esempio con strutture nascoste e modificatore static.....	127
Esercizi.....	131
Array dinamico di struct.....	132
C Tutorial.....	132
Esempio di array dinamico di struct.....	132

Esercizi.....	140
Puntatore a void [approfondimento].....	144
C Tutorial.....	144
Esempio su puntatori a void e allocazione dinamica.....	144
File.....	147
C Tutorial.....	147
Esempi.....	147
File access basics.....	147
Read CSV file data into an array of structs.....	149
Esercizi.....	151
Algoritmi.....	152
Algoritmi di ordinamento.....	152
Bubble sort.....	152
Esercizi.....	153
Algoritmi di ricerca.....	154
Binary search.....	154
Exercises.....	154
Esempi di ricerca binaria.....	155
Memoria statica.....	155
Memoria dinamica.....	155
Algoritmi ricorsivi.....	158
Recursion.....	158
Recursive binary search.....	158
Una questione di stile.....	159
Esercizi avanzati.....	160
Array.....	160
Algoritmi di ordinamento.....	163
Matrici e funzioni.....	175
Funzioni, ricorsione, array e matrici.....	179
Problemi vari.....	182
Documentazione di progetto finale.....	185
Esempi di progettazione.....	187
Array paralleli - Città e PM10.....	187
Struct - Vendita di prodotti.....	193
Complessità degli algoritmi.....	197
C Language Manuals.....	198
Per le OI.....	199
OOP in C language.....	200
Tutorial.....	200
Appendici.....	201
A - C course videos.....	202
B - Restituzione dell'indirizzo di memoria di un array da una funzione.....	207
C - Passaggio dei parametri per valore e per indirizzo.....	212
Uso dei puntatori nelle funzioni per la gestione dell'array di int.....	212
Uso di array nelle funzioni per la gestione dell'array di int e di stringhe.....	218

D - OLD Esempio d'uso dei file.....	227
Bibliografia e sitografia.....	239

Problemi, programmi e algoritmi¹

Generalmente la necessità di soddisfare dei bisogni, come ad esempio preparare una pietanza, porta il soggetto a cercare una soluzione operando in autonomia o chiedendo aiuto ad una persona esperta che svolge il compito al posto suo, in cambio di un compenso. Chi opera per risolvere il problema viene chiamato **risolutore**, mentre chi chiede la risoluzione del problema è il **cliente** o **committente**.

Per risolvere il problema il **risolutore** lo analizza ponendosi una serie di domande che gli permettono di comprendere chiaramente cosa fare:

- Chi sono tutte le persone coinvolte?
- Qual è l'obiettivo finale?
- Cosa ho a disposizione?
- Cosa invece mi serve?
- Qual è il procedimento da seguire?
- Ci sono dei casi limite?
- Come faccio a sapere se ho raggiunto l'obiettivo?

Così nel caso della pietanza da preparare, le domande che bisogna porsi sono le seguenti:

- Ho capito bene che pietanza devo fare?
- Ho tutti gli ingredienti che mi servono?
- Ho la ricetta?
- So come verificare se la cottura e la preparazione sono ottimali?

Dopo aver analizzato il problema, trovando una risposta a tutte le domande precedenti e, avendo indicato qual è il procedimento da seguire, l'**esecutore** risolve il problema o, nel caso della pietanza, la prepara effettivamente.

Quindi, un problema nasce da un bisogno che generalmente porta alla creazione di qualcosa che prima non c'era. Gli attori coinvolti in questo atto creativo sono il **cliente** che ha il problema, il **risolutore** che si prende la responsabilità di risolverlo e l'**esecutore** che, seguendo i passi definiti dal **risolutore**, produce quello di cui il **cliente** ha bisogno.

Molti problemi della vita quotidiana ammettono come soluzione un programma che può essere eseguito da computer. In questo caso il **cliente** può essere una **persona fisica** o una **società** che chiede la soluzione di un problema che può essere automatizzato. La figura del **risolutore** in questo caso è l'**analista/programmatore** che, confrontandosi con il **cliente**, raccoglie tutte le **specifiche** necessarie per **formalizzare il problema**. La fase di **raccolta delle specifiche** è fondamentale affinché il **programmatore** possa **trasformarle in un programma** scritto in uno specifico linguaggio di programmazione. Il programma così ottenuto è il procedimento che viene fornito all'**esecutore** che, in questo caso è il **computer**. Un computer, avendo a disposizione le istruzioni precise da svolgere, è in grado di risolvere il problema.

Se ad esempio, un cliente volesse un programma in grado di calcolare la media mensile dei guadagni del suo negozio, la raccolta delle specifiche dovrebbe prevedere le seguenti domande:

¹ Estratti delle presentazioni [Il big bang delle idee](#) e [Dal problema al programma](#) del Prof. [Fabio Biscaro](#)

- Chi sono gli attori coinvolti? → Il negoziante che inserisce il guadagno delle vendite giornaliere per tutto il mese.
- Quali sono gli input? → Le vendite giornaliere
- Quali sono gli output? → La media mensile delle vendite
- Quali sono i passi da eseguire? →
 - Inserimento giornaliero delle vendite
 - Se una vendita è negativa segnalo l'errore
 - Accumulo delle vendite
 - Calcolo della media mensile come divisione della somma accumulata per il numero di giorni del mese
 - Visualizzazione della media
- Quali sono i casi limite? → Non si devono poter inserire valori negativi nelle vendite
- Come faccio a sapere se funziona? (Test) → Inserimento dei numeri di cui si conosce già qual è la media per verificarne l'esattezza. Verifica del caso limite inserendo dei numeri negativi per vedere se viene impedito questo input.

Una volta individuate tutte le specifiche, può essere necessario **descrivere l'algoritmo** con diversi formalismi, come ad esempio un **flow-chart** o un **caso d'uso**, che ci permette di analizzare la logica di funzionamento prima ancora di procedere con la verifica, individuando errori logici che potrebbero essere difficili da trovare in un programma.

Infine l'**algoritmo descritto dal flow-chart** o dal **caso d'uso** viene **tradotto in un programma** scritto nel linguaggio di programmazione scelto, effettuando tutti i test previsti per essere certi che funzioni correttamente.

La consegna di un programma è però difficilmente la conclusione del nostro lavoro; frequentemente un cliente ci contatta nuovamente per chiedere l'aggiunta di caratteristiche all'applicazione o, nel caso in cui non fossimo stati sufficientemente attenti nella fase di testing, per richiedere la correzione di un errore che ha riscontrato nell'applicazione.

Un [algoritmo](#) è una **sequenza finita** di istruzioni che definiscono in modo chiaro e **non ambiguo** le operazioni da eseguire per raggiungere un risultato.

Un algoritmo, per essere considerato tale, deve rispettare le seguenti caratteristiche:

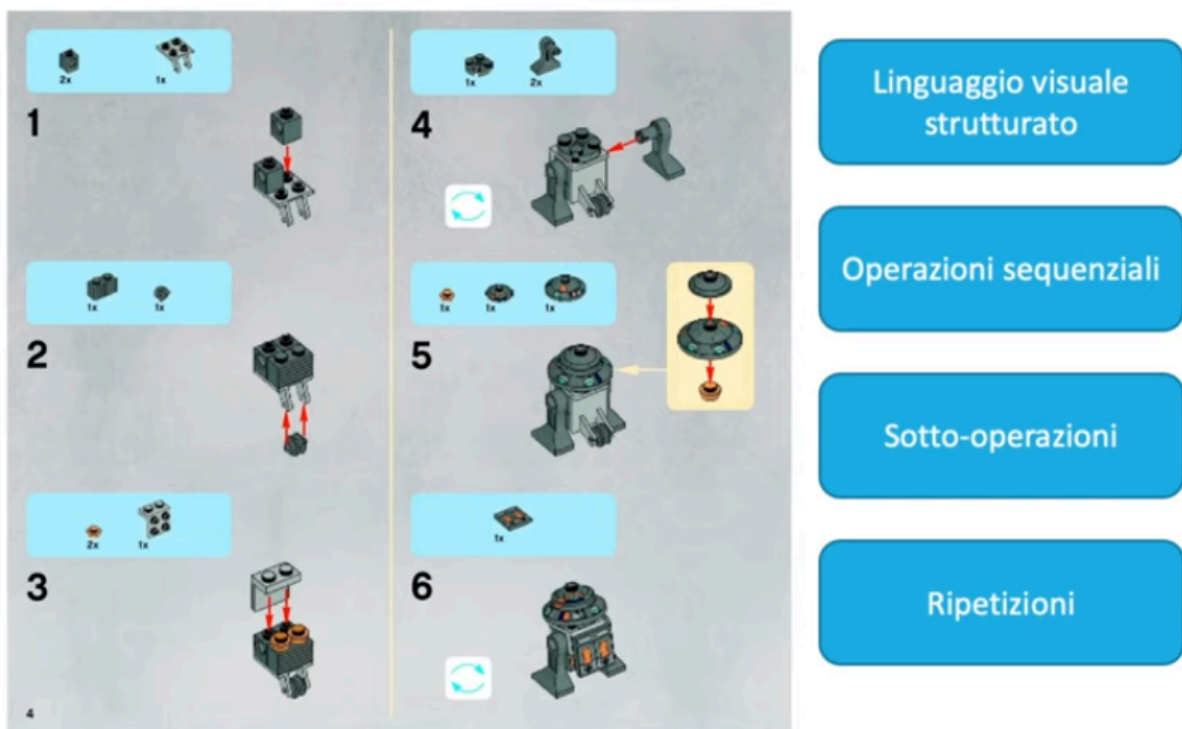
- deve essere **finito** in tempo e in spazio, cioè deve essere costituito da un numero finito di istruzione e deve sempre terminare;
- deve essere **deterministico**, cioè partendo dagli stessi dati in ingresso, si devono ottenere i medesimi risultati in uscita;
- deve essere **non ambiguo**, cioè le operazioni non devono poter essere interpretate dall'esecutore in modi differenti;
- deve essere **generale**, cioè deve essere applicabile a tutti i problemi della stessa classe a cui si riferisce.

Un qualsiasi procedimento che non rispetti anche solo una delle caratteristiche sopra elencate non può essere definito un algoritmo.

Come già indicato, molti problemi della vita reale sono risolvibili tramite un algoritmo e, quindi, tramite un programma, ma non tutti i problemi sono riconducibili ad un algoritmo in quanto le loro specifiche non portano ad una soluzione che rispetti le caratteristiche indicate precedentemente.

Di seguito vengono rappresentati due algoritmi che possono presentarsi nella vita reale, entrambi derivati dal [corso di Informatica di base del primo anno](#) tenuto del [Prof. Fulvio Corno](#) presso il [Politecnico di Torino](#).

Il primo permette di costruire un robottino usando i lego e segue tutti i principi elencati in precedenza: è finito, è deterministico, non è ambiguo ed è generale per il compito da svolgere.



Il secondo usa un linguaggio verbale che, pur sembrando chiaro, porta con sé l'ambiguità del linguaggio naturale con conseguenze imprevedibili se si segue un pensiero rigorosamente algoritmico come mostrato nel primo esempio.

È quindi necessario analizzare con cura un problema per eliminare qualsiasi ambiguità presente e renderlo effettivamente un algoritmo.

Wife : Honey, please go to the super market and get 1 bottle of milk. If they have bananas, bring 6.

He came back with 6 bottles of milk.

Wife: Why the hell did you buy 6 bottles of milk?!?!

Husband (confused): BECAUSE THEY HAD BANANAS.

He still doesn't understand why his wife yelled at him since he did exactly as she told him.

La programmazione strutturata

La [programmazione strutturata](#) è un [paradigma di programmazione](#) che ha permesso, alla fine degli anni '60, di ripensare la scrittura del codice in modo tale che fosse più **leggibile** e **manutenibile**.

Fino al ripensamento delle tecniche di programmazione seguendo il paradigma strutturato, la [programmazione procedurale](#) era costituita da istruzioni che permettevano di saltare in qualsiasi punto del codice, senza che fosse usato un criterio chiaro e definibile che rendesse semplice la sua lettura anche ad un programmatore che non aveva scritto il programma. La difficile lettura rendeva quasi del tutto impossibile mantenere e modificare un codice scritto da altri e questo, nel tempo, si è dimostrato un approccio fallimentare per molti progetti informatici. Il problema divenne così rilevante che fu coniata una nomenclatura dispregiativa per questo tipo di approccio alla programmazione, [spaghetti code](#).

Per superare il problema dello *spaghetti code* molti tra i più accreditati programmatori dell'epoca decisero di sviluppare tecniche e regole di scrittura del codice che rendessero semplice la lettura e la modifica dello stesso. Durante il forte fermento nato all'interno della comunità matematica ed informatica per trovare una soluzione allo *spaghetti code*, gli informatici [Corrado Böhm](#) e [Giuseppe Jacopini](#) enunciarono nel 1966 il loro celebre teorema, il [Teorema di Böhm-Jacopini](#) che afferma quanto segue.

Qualunque algoritmo può essere implementato in fase di programmazione utilizzando tre sole strutture dette **strutture di controllo**:

1. la **sequenza**;
2. la **selezione**;
3. l'**iterazione**.

Queste tre strutture devono essere applicate in modo gerarchico alle istruzioni elementari che compongono l'algoritmo.

Gli informatici Böhm e Jacopini fornirono una modalità per riscrivere qualsiasi programma che utilizza la struttura del salto incondizionato (*GOTO*) usando le tre strutture di controllo di **sequenza**, **selezione** e **iterazione**.

Linguaggi di programmazione strutturati

Un **linguaggio di programmazione** si definisce **strutturato** se nelle sue regole sintattiche e semantiche ammette le tre strutture di controllo descritte dal **Teorema di Böhm-Jacopini** e rispetta i tre requisiti seguenti.

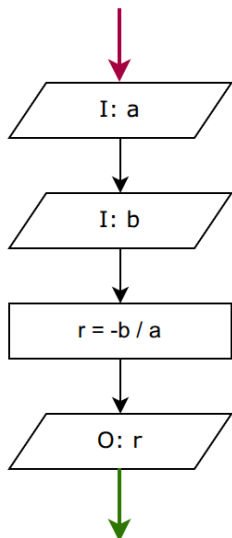
1. Il linguaggio deve essere **completo** e quindi deve offrire la struttura di **sequenza**, almeno una struttura di **selezione** e almeno una struttura di **iterazione**.
2. Il linguaggio deve prevedere che ciascuna struttura di controllo (*sequenza, selezione e iterazione*) abbia un **singolo punto di ingresso** e un **singolo punto di uscita**. Questo vuol dire che ogni struttura di controllo, costituita dalle sue istruzioni elementari, deve essere considerata come una singola *macro-istruzione* con un unico identificabile punto di ingresso ed un unico identificabile punto di uscita.
3. Il linguaggio deve permettere di **comporre** le strutture di controllo (*sequenza, selezione e iterazione*) **ricorsivamente**, cioè inserire all'interno di una struttura di controllo un'altra struttura.

Gli ultimi due requisiti permettono di creare flussi di controllo costituiti da strutture di controllo completamente isolate dalle altre, incapaci di interferire tra loro. Questa condizione è indispensabile per applicare metodologie progettuali quali la [progettazione top-down e bottom-up](#).

Le strutture di controllo

Le tre strutture di controllo definite dal **Teorema di Böhm-Jacopini**, corredate dei tre requisiti dei linguaggi strutturati, sono rappresentabili usando un linguaggio di modellazione grafico come il [diagramma di flusso](#) o *flow-chart*.

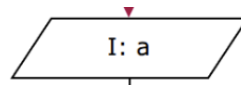
Sequenza



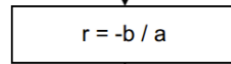
La struttura di controllo di **sequenza** permette di eseguire successivamente una serie di istruzioni, una di seguito all'altra.

Il diagramma di flusso mostra la struttura di controllo **sequenza** costituita dalle istruzioni elementari di

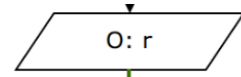
- input



- esecuzione



- output

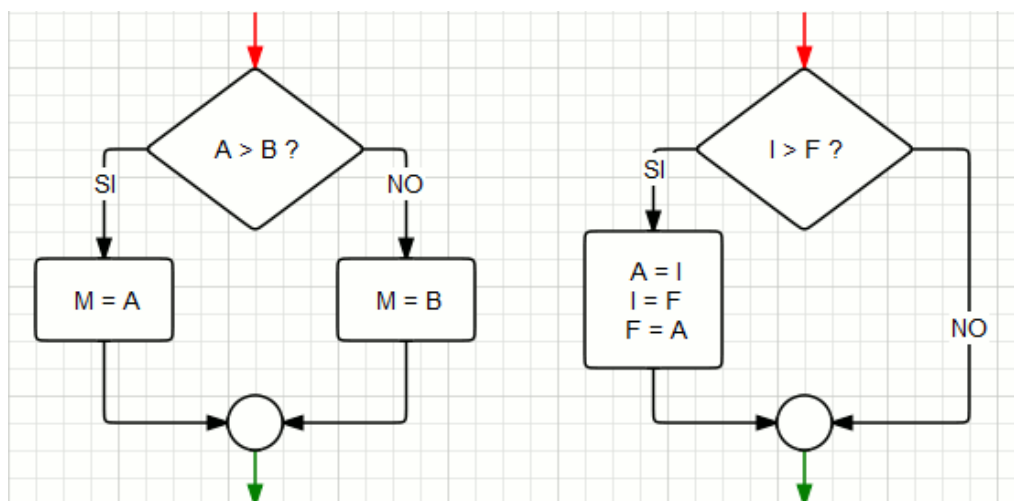


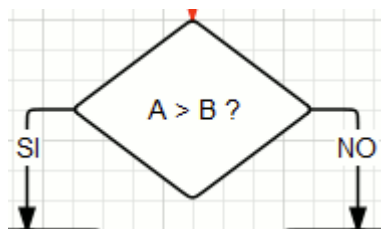
La **freccia rossa** ↓ indica l'unico punto di ingresso della struttura di controllo **sequenza**, mentre la **freccia verde** ↓ indica l'unico punto di uscita.

La **freccia rossa** ↓ e la **freccia verde** ↓ identificano una singola *macro-istruzione di sequenza* che può essere innestata all'interno delle altre strutture di controllo.

Selezione

La struttura di controllo di **selezione** permette di eseguire esclusivamente un gruppo di istruzioni, tralasciando quello alternativo. La scelta di quale gruppo di istruzioni eseguire dipende dal valore assunto dal test iniziale.





Entrambe le strutture di **selezione** mostrate sopra sono costituite da un test da cui si dipanano due percorsi distinti e mutualmente esclusivi.

Durante una esecuzione viene seguito uno solo dei due percorsi, il percorso alternativo non sarà mai eseguito in quella specifica esecuzione dell'algoritmo.

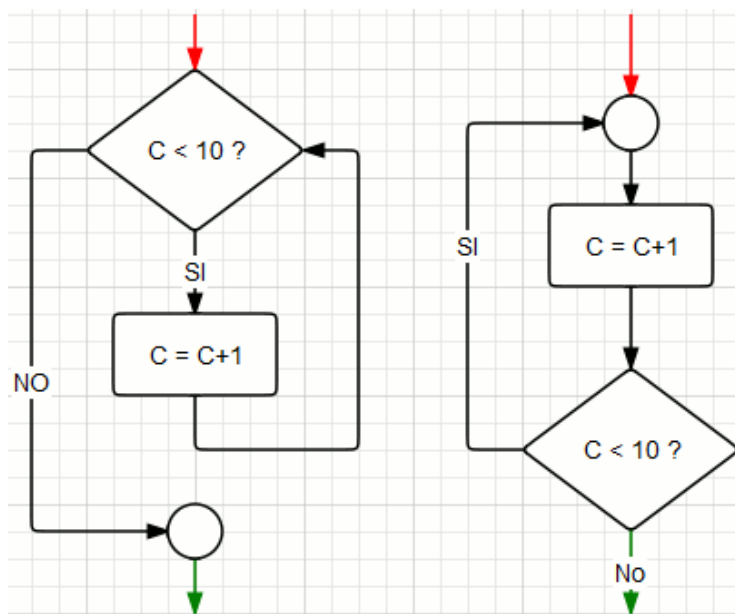
Come mostrato nei due esempi, la **selezione** può o meno avere delle *macro-istruzioni* nel ramo NO (*falso*), mentre deve sempre avere delle *macro-istruzioni* nel ramo SI (*vero*).

La **freccia rossa** ↓ indica l'unico punto di ingresso della struttura di controllo, mentre la **freccia verde** ↓ indica l'unico punto di uscita.

La **freccia rossa** ↓ e la **freccia verde** ↓ identificano la **selezione** come una singola *macro-istruzione di selezione* che può essere innestata all'interno delle altre strutture di controllo.

Iterazione

La struttura di controllo di **iterazione** permette di eseguire un blocco di istruzioni più volte, fino a quando il test risulta vero. Questa è una differenza sostanziale rispetto alla **selezione** perché nell'**iterazione** non viene selezionato un percorso ignorando l'altro, il suo scopo è invece quello di ripetere più volte la *macro-istruzione* contenuta nel suo ramo SI (*vero*).



Nello specifico il diagramma di flusso a sinistra identifica una **iterazione precondizionale**, che prevede la verifica del test prima di svolgere la *macro-istruzione* presente nel ramo vero. Nel caso in cui il test dell'**iterazione precondizionale** risulti subito falso, la *macro-istruzione* nel ramo vero non viene mai eseguita.

Nel diagramma di flusso a destra è invece mostrata una **iterazione postcondizionale** che prevede la verifica del test dopo aver eseguito la *macro-istruzione* presente nel suo ramo vero. Nel caso in cui il test dell'**iterazione postcondizionale**

risulti subito falso la *macro-istruzione* presente nel ramo vero viene sempre eseguita almeno una volta perché il test è successivo alla *macro-istruzione* del ramo vero.

Un **linguaggio strutturato** non deve necessariamente gestire entrambi i costrutti iterativi, infatti combinando opportunamente le diverse strutture di controllo è possibile ottenere i medesimi comportamenti di una **iterazione precondizionale** o **postcondizionale**.

Documentazione di progetto iniziale

Per la risoluzione dei problemi presentati durante il corso viene proposta una iniziale struttura della documentazione di progetto che si rifà a quanto visto nei paragrafi precedente. Questa iniziale struttura permette di analizzare un problema e di definire le sue specifiche in modo guidato, permettendo di raggiungere una migliore capacità di analisi che consentirà successivamente di utilizzare formalismi più professionali.

Consegna - Il testo del problema è la proposta di una situazione reale in cui un committente ti chiede di sviluppare un'applicazione per risolvere un problema. Esso rappresenta in pratica i requisiti che il tuo sistema dovrà soddisfare.

Per essere certo che i requisiti siano completi devi analizzare il problema e, se necessario, completare i requisiti iniziali intervistando il committente o aggiungendone di nuovi dopo aver analizzato e compreso il problema.

Dato che il tuo sistema deve funzionare nel mondo reale devi essere certo di aver previsto tutti i percorsi necessari affinché il sistema funzioni, anche nel caso qualcosa andasse storto.

Titolo del progetto - Ogni progetto che farai sarà degno di un nome, inventane uno rappresentativo della tua applicazione.

Chi usa l'applicazione - L'utilizzatore dell'algoritmo può essere qualsiasi entità, un umano, un animale o un altro programma. Molto spesso i nostri algoritmi saranno avviati da un attore umano, ma non è detto che sia sempre così, è necessario che ti soffermi a pensarci perché in futuro questo aspetto sarà importante.

Obiettivo principale dell'algoritmo - Descrizione sintetica dell'obiettivo principale dell'algoritmo. Serve per individuare qual è effettivamente la richiesta del committente. Se non si capisce qual è l'obiettivo è impossibile fornire un algoritmo che soddisfi la richiesta.

Strutture dati - Elenca nella forma che preferisci le variabili usate indicando il nome, il tipo, la descrizione della sua funzione, se è di input (I), di output (O) o di lavoro (L) e il relativo dominio di validità.

Analisi di progetto - Rispondi alle seguenti domande per analizzare in modo [metacognitivo](#) la tua progettazione e implementazione dell'applicazione durante la sua creazione.

1. Gli input dell'applicazione sono stati adeguatamente controllati? Individua correttamente il dominio di validità degli input e descrivi brevemente quali controlli hai previsto di fare. Se necessario aggiungi o modifica i controlli per rispettare i corretti domini di validità.
2. L'applicazione è stata separata in moduli diversi in modo che sia ben definita la competenza di ognuno di essi? Descrivi brevemente qual è la competenza generale di ogni modulo. Se il modulo ha troppe competenze sposta le funzioni nei moduli corrispondenti, eventualmente creandone dei nuovi.
3. Tutte le funzioni all'interno di un modulo sono fortemente coese e debolmente accoppiate? Descrivi queste caratteristiche relativamente alla tua applicazione. Se ci sono forti dipendenze tra funzioni riprogetta in modo che ci sia un basso

accoppiamento. Se sono presenti delle funzioni che non rispondono alla competenza principale del modulo, spostale nel modulo corretto.

4. Sei sicuro che ogni funzione rispetti il principio di singola responsabilità? Se c'è più di una ragione per modificarla probabilmente ha troppe responsabilità e deve essere separata in più funzioni. Se necessario sistema la tua applicazione.

Casi d'uso

Caso d'uso principale - In questa sezione si descrive come viene usata l'applicazione dal punto di vista di un utente che ha fatto una richiesta al sistema. Un caso d'uso cattura la sequenza di eventi *visibili* attraverso cui il sistema deve procedere per rispondere ad una singola richiesta dell'utente. I casi d'uso sono derivati dai colloqui effettuati con gli [stakeholder](#) e, per loro natura sono destinati a cambiare frequentemente, quindi è bene mantenerli semplici. Ogni caso d'uso presenta almeno un *caso principale* che descrive la risposta del sistema quando tutto procede correttamente. Per ogni *Caso d'uso principale* è possibile prevedere anche degli esempi di ciò che deve essere visualizzato e inserito.

Casi d'uso alternativi - Spesso nei *Casi d'uso principali* sono presenti dei *Casi d'uso alternativi*, che si verificano quando il *Caso d'uso principale* può presentare dei problemi da gestire. Nei *Casi d'uso alternativi* è necessario indicare quando si verificano e cosa bisogna fare per recuperare la situazione anomala. Anche per i *Casi d'uso alternativi* si prevedono degli esempi di ciò che viene visualizzato dal programma.

Diagramma di flusso (flow-chart) - Rappresentazione grafica dell'algoritmo risolutivo. Questa modalità di rappresentazione dell'algoritmo usa un linguaggio di programmazione grafico ed è realizzato dall'analista/programmatore. L'esecutore è lo stesso programmatore che simula il comportamento dell'elaboratore. Nelle fasi iniziali della programmazione questa tecnica è utile per migliorare le capacità logiche e analitiche.

Codifica - Implementazione in un linguaggio di programmazione dell'algoritmo risolutivo. Questa modalità di rappresentazione dell'algoritmo usa un linguaggio di programmazione interpretabile da un computer che diviene l'esecutore. Il programmatore è il realizzatore del programma e le capacità logiche e analitiche richieste per la sua stesura sono le stesse di quelle richieste per un diagramma di flusso.

Screenshot dei test programmati (se richiesti) - Screenshot dei test programmati dimostrativi del funzionamento del programma.

Sequenza, selezione, iterazione

C Tutorial

1. C Introduction by [Programiz](#)
 - a. [Your First C Program](#)
 - b. [C Keywords and Identifiers](#)
 - c. [C Variables, Constants and Literals](#)
 - d. [C Data Types](#)
 - e. [C Input Output \(I/O\)](#)
 - f. [C Comments](#)
 - g. [C Programming Operators](#)
 - h. [C Introduction Examples](#)
2. C Flow Control by [Programiz](#)
 - a. [C if...else Statement](#)
 - b. [C for Loop](#)
 - c. [C while and do...while Loop](#)
 - d. [C break and continue](#)
 - e. [C switch Statement](#)
 - f. [C Ternary Operator](#)
 - g. [C goto Statement](#) ← IL MALE!
 - h. [C Control Flow Examples](#)
3. VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)
 - a. [Basics of variables and types](#)
 - b. [printf Basics](#)
 - c. [scanf Basics](#)
 - d. [Comments | C Programming Tutorial](#) ← l'argomento è spiegato usando gli array che vedremo più avanti, ma cerca di comprendere come scrivere un commento, senza preoccuparti della complessità del codice
 - e. [How To Print The % Character Using printf\(\)](#)
 - f. [pow\(\) function](#)
 - g. [Find A Square Root Using sqrt\(\) sqrtf\(\) sqrtl\(\)](#)
 - h. [Arithmetic Operations](#)
 - i. [postfix vs. prefix increment operator](#)
 - j. [Assignment Operators](#)
 - k. [if statements](#)
 - l. [Relational Operators](#)
 - m. [Logical Operators](#)
 - n. [If Statement Best Practice](#)
 - o. [while loop C](#)
 - p. [do while loop](#)
 - q. [break and continue statements](#)
 - r. [for loops](#)
 - s. [switch Statements](#)
 - t. [Convert A For Loop To A While Loop](#)
 - u. [Ternary Operator](#)

- v. [Type Casting](#)
- w. [Main Function Return Value](#)
- x. [How True and False Boolean Values Work In C](#)
- y. [Random Number Generation](#)
- z. [Absolute Value Functions abs\(\) And fabs\(\)](#)
- aa. [Floating-Point Number Classification Functions](#)
- bb. [How To Clear The Input Buffer](#) ← *l'argomento è spiegato usando array e stringhe che vedremo più avanti, ma cerca di comprendere la modalità di pulizia del buffer di input, senza preoccuparti della complessità del codice*

Esercizi

1. Progettare un'applicazione che data l'età di una persona, visualizzare se è maggiorenne o minorenne.
2. Progettare un algoritmo che calcoli la somma di tre numeri interi.
3. Progettare un algoritmo che calcoli l'area di un triangolo equilatero fornendo solo la misura del lato.
4. Progettare un algoritmo che calcoli l'area di un triangolo equilatero.
5. Progettare l'algoritmo che stampi a video la seguente figura:

6. Progettare un algoritmo che legga un valore intero e visualizzi il valore intero precedente e il successivo.
7. Progettare un algoritmo che, dati due numeri interi *dividendo* e *divisore*, calcoli il quoziente e il resto della divisione intera tra *dividendo* e *divisore* (per ottenere il resto della divisione tra interi usare l'operatore %).
8. Progettare un algoritmo che, date le età di tre persone, calcoli l'età media.
9. Progettare un algoritmo che preveda l'inserimento della misura di un lato e visualizzi i seguenti valori:
 - a. l'area del quadrato avente il lato pari a quello inserito;
 - b. l'area del rettangolo avente la base pari al lato inserito, e l'altezza pari alla metà del lato inserito;
 - c. l'area del triangolo rettangolo avente l'altezza pari al lato inserito, e la base pari ad $\frac{1}{3}$ del lato inserito.
10. Progettare un algoritmo che, dato il prezzo di un prodotto, calcoli il prezzo scontato del 20%.
11. Progettare un algoritmo che, ricevuto in input un orario attraverso le sue tre componenti (ore, minuti e secondi), ne calcoli il valore totale in secondi. La visualizzazione del risultato deve essere come mostrato nell'esempio di seguito.
Ore: 2 - Minuti: 56 - Secondi: 3
Secondi: 10563
12. Progettare un programma che legga in input un valore corrispondente a un numero di secondi e lo trasformi in ore, minuti e secondi. La visualizzazione del risultato deve essere come mostrato nell'esempio di seguito.
Secondi: 10563
Ore: 2 - Minuti: 56 - Secondi: 3

13. La Prof.ssa Maffucci, che è un po' tappina, vuole sapere se nella tua classe c'è qualcuno più alto di lei e quindi ti chiede di progettare un algoritmo che visualizzi l'altezza massima in centimetri di tutti gli studenti della tua classe.
14. Il Professore di Scienze Motorie fa partecipare tutta la tua classe ad una competizione di velocità. Devi progettare un algoritmo che visualizzi il tempo in secondi del partecipante che ha vinto la gara.
15. Progettare un algoritmo che calcoli e visualizzi le radici nel campo dei numeri reali di un'equazione di secondo grado, dati in input i coefficienti dell'incognita di primo e secondo grado e il termine noto.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Si rammenta che nel campo dei numeri reali le soluzioni sono sempre due, eventualmente coincidenti, a seconda del valore assunto dal determinante, e la formula è mostrata di lato.

Nel caso il coefficiente di secondo grado sia pari a zero si visualizzi comunque la soluzione dell'equazione insieme ad un messaggio che segnali l'anomalia. Nel caso non sia possibile determinare le radici nel campo dei numeri reali, si visualizzi un messaggio opportuno.

16. Realizzare un algoritmo per la simulazione di un circuito elettrico, composto da un generatore di tensione e da due resistenze in parallelo (R1 e R2), che svolga le seguenti funzioni:
 - a. Legga i valori, in Ohm, delle resistenze R1 ed R2
 - b. Legga il valore, in Volt, del generatore di tensione (V1)
 - c. Stampi una rappresentazione grafica del circuito elettrico (vedi l'esempio di stampa)
 - d. Calcoli il valore della resistenza equivalente al parallelo e delle correnti passanti per le due resistenze
 - e. Stampi i dati iniziali e i valori calcolati.

Esempio di stampa con caratteri

```
-----
|                                     |
|                                     |
|               -----              |
|               I1 |               I2 |
|               ---              ---  |
|               R1 | |             R2 | |
V1 -           | |               | |
-----       | |               | |
|               ---              ---  |
|               |                 |   |
|               -----              |
|                                     |
```

V1= 3.4 Volt R1=230 Ohm R2=180 Ohm R1//R2= 100,97 Ohm
I= 0,03367 A I1=0,01478 A I2=0,01888

17. Progettare un algoritmo che prenda in input un numero intero e visualizzi la scritta **pari** se il numero è pari, e **dispari** se il numero è dispari.
18. Progettare un algoritmo che, dati due numeri interi, calcoli il risultato e il resto della divisione intera tra i due numeri.
19. Indicare per quali valori di a, b e c si ottiene come output Il primo coefficiente è negativo eseguendo il seguente frammento di codice. Si descriva l'algoritmo utilizzando i flow chart modificandolo in modo tale che l'output corrisponda esattamente ai valori inseriti.

```
if (a > 0) {  
    if (b > 0) {  
        if (c > 0) {  
            cout << "Sono tre coefficienti positivi";  
        } else {  
            cout << "Il primo coefficiente è negativo";  
        }  
    }  
}
```

20. Scrivere l'algoritmo che, letto in input un valore numerico, dica se è positivo o negativo.
21. Scrivere l'algoritmo che, letto in input un numero intero, dica se è pari o dispari.
22. Scrivere un algoritmo che, letti in input due numeri interi, verifichi se il primo è multiplo del secondo.
23. Scrivere un algoritmo che, dato un numero intero in input, visualizza il suo doppio se è pari, il triplo se è dispari.
24. Scrivere l'algoritmo che, dati due numeri, calcoli la somma se sono entrambi positivi, il prodotto altrimenti.
25. Scrivere un algoritmo che, dato il prezzo di un prodotto, applichi uno sconto del 12% se il prezzo è inferiore a € 30,00, del 25% altrimenti.
26. Scrivere l'algoritmo che, dati base e altezza di un triangolo, calcoli l'area se sono entrambi positivi, oppure stampi il messaggio "Valori di input errati".
27. Scrivere l'algoritmo che, presi in input gli estremi a e b di un intervallo e un valore x, visualizzi il messaggio "Il valore è interno all'intervallo" se $a \leq x \leq b$, altrimenti "Il valore è esterno all'intervallo".
28. Scrivere un algoritmo che, preso in input un voto, dica se è corretto (compreso tra 1 e 10).

29. Scrivere un algoritmo che, dato il consumo di acqua di un utente, espresso in m^3 , calcoli l'importo della bolletta, sapendo che ogni bolletta comprende una quota fissa di 20 euro e una quota variabile di 2,50 euro/ m^3 per i primi 100 metri cubi d'acqua, di euro 4,00/ m^3 per i metri cubi in eccesso.
30. Scrivere l'algoritmo per controllare la correttezza di un orario ricevuto in ingresso, attraverso tre diversi input: h, m e s (nel formato 24 ore).
- a. ore, int, numero di ore, I, valore numerico e $0 \leq \text{ore} \leq 23$
 - b. minuti, int, numero di minuti, I, valore numerico e $0 \leq \text{minuti} \leq 59$
 - c. secondi, int, numero di secondi, I, valore numerico e $0 \leq \text{secondi} \leq 59$
31. Scrivere un algoritmo che determini il prezzo di ingresso al cinema considerando che clienti con un'età compresa tra 18 e 65 anni pagano il prezzo pieno, mentre per età diverse da queste il prezzo è la metà del prezzo pieno. Si effettuino i dovuti controlli sui valori immessi in input.
32. Scrivere l'algoritmo per controllare la correttezza di una data ricevuta in ingresso attraverso tre diversi input: giorno, mese e anno. Tutti gli input devono essere numerici. Si effettuino i dovuti controlli sui valori immessi in input.
33. Progettare un programma che dati in input due orari della stessa giornata visualizzi la differenza in ore, minuti e secondi. Effettuare i controlli sui valori immessi.
- Esempio:
- ```
Ora 1: 10
Minuti 1: 14
Secondi 1: 18
Ora 2: 5
Minuti 2: 9
Secondi 2: 2
Differenza: 5:5:16
```
34. Realizzare un programma che fornite due date in input, visualizzi il numero di giorni che intercorrono tra loro. Ad esempio:
- ```
Data antecedente: 10/08/2018      Data successiva: 08/01/2022

Giorni che intercorrono fra le due date: 1243
```
- Per ogni data prevedere l'inserimento distinto di giorno, mese, anno. Non è previsto che la prima data inserita debba necessariamente essere l'antecedente. Non si considerino gli anni bisestili, i mesi si suppongono tutti di 30 giorni e gli anni compresi tra il 1970 e l'anno corrente. Si controlli la correttezza delle date inserite.
35. Lo spazio espresso in metri di frenata di un'automobile è stimato mediante la seguente formula, supponendo che il tempo di reazione del guidatore sia pari ad 1 secondo: $\text{spazio} = \text{velocita}^2 / (250 * \text{COEFFICIENTE})$ dove *velocita* è la velocità in km/h, e *COEFFICIENTE* è un coefficiente relativo alle condizioni stradali, come indicato dalla seguente tabella:

Condizioni stradali	COEFFICIENTE
Asfalto ruvido	0,6
Asfalto liscio	0,5
Asfalto bagnato	0,4
Asfalto ghiacciato	0,1

Progettare un algoritmo che calcola lo spazio di frenata a partire dalla velocità data in input e dal coefficiente sulle condizioni stradali. Usare lo switch-case per risolvere l'algoritmo e usare le costanti o le macro nel codice per i valori dei coefficienti.

36. La Prof.ssa Maffucci assegna i livelli di valutazione ai propri studenti in base alla seguente tabella dei punteggi:

Punteggio	Valutazione
0-10	4
11-20	5
21-30	6
31-40	7
41-50	8

Supponendo che sia i punteggi, sia i voti siano sempre interi, progettare un algoritmo che consente di determinare il livello di valutazione a partire dal punteggio.

37. Gli abbonamenti della GTT di Torino possono riguardare zone diverse (urbano - 1, suburbano - 2, urbano+suburbano - 3) e possono essere settimanali (S), mensili (M) o annuali (A). I costi sono quelli indicati nella seguente tabella:

Zona\Durata	S	M	A
1	12,00 €	38,00 €	310,00 €
2	9,80 €	35,50 €	319,50 €
3	15,70 €	56,50 €	508,50€

Gli abbonamenti per gli studenti hanno una riduzione del 50%.

Progettare un algoritmo che calcola il costo dell'abbonamento a partire dal tipo, dalla durata e se ad acquistarlo è uno studente o meno.

38. Progettare un algoritmo che accetti tre valori numerici interi e li visualizzi in ordine crescente.
39. Progettare un algoritmo che richieda l'inserimento di una temperatura in gradi Celsius e visualizzi un messaggio come indicato nella seguente tabella:

Temperatura (t)	Messaggio
-----------------	-----------

$t \geq 30$	"Molto caldo"
$20 \leq t < 30$	"Caldo"
$10 \leq t < 20$	"Ideale"
$0 \leq t < 10$	"Freddo"
$t < 0$	"Molto freddo"

40. Progettare un algoritmo che calcoli le radici nel campo reale di un'equazione di secondo grado, dati in input i coefficienti a e b e il termine noto c .
41. Progettare un algoritmo che visualizzi il maggiore fra tre numeri dati in input.
42. Progettare un algoritmo in grado di stabilire se un anno fornito in input è o meno bisestile.
43. Progettare un algoritmo in grado di visualizzare il numero di giorni di un mese dato come input. Il mese dovrà essere inserito come valore numerico.
44. Progettare un algoritmo che svolga le seguenti funzioni:
- Legga due date (vedi punto A)
 - Stampi il numero di giorni che intercorrono fra la data antecedente e la data successiva secondo le indicazioni (vedi punto B)
- A) Per ogni data prevedere l'inserimento distinto di giorno, mese, anno.
Non è previsto che la prima data inserita sia necessariamente l'antecedente.
Controllare la correttezza delle date.
Gestire gli anni bisestili.
Gestire i diversi giorni dei mesi.
Gli anni devono essere compresi nell'intervallo [1970-anno corrente]

B) Esempio di output

Data antecedente: 10/08/2012 Data successiva: 08/01/2016

Giorni che intercorrono fra le due date: XXXX

45. Progettare un algoritmo che chieda in input tre valori interi compresi nell'intervallo [10-1000]. L'algoritmo deve verificare se almeno due dei tre numeri condividono la cifra meno significativa, e in questo caso dovrà visualizzare la cifra condivisa. Nel caso non ci sia alcun numero che condivide la cifra meno significativa si dovrà visualizzare un messaggio appropriato.
- Esempi:
- Input: 41, 22, 71 → Output: 1
- Input: 23, 32, 42 → Output: 2
- Input: 57, 37, 17 → Output: 7
- Input: 14, 53, 98 → Output: "Nessun valore condivide l'ultima cifra"
- Input: 9, 99, 999 → Output: "Una delle cifre non si trova nell'intervallo [10-1000]"
46. Progettare un algoritmo che prenda in input un numero intero, positivo o negativo, e visualizzi il numero come parola se è compreso nell'intervallo [-9, +9], mentre visualizzi la scritta "Other" diversamente. Si provi ad usare il costrutto switch-case.

47. Realizzare un programma in grado di calcolare l'altezza media degli studenti di una classe, dando in input il numero di allievi della classe.
48. Realizzare un programma in grado di calcolare l'altezza media degli studenti di una classe, senza fornire inizialmente il numero di allievi della classe.
49. Progettare un algoritmo che legga una sequenza di valori numerici fino alla lettura di un valore 0 e scriva quanti valori sono stati letti e la loro somma.
50. Progettare un algoritmo che, dato un valore numerico numero, legga n valori e conti quanti di essi sono maggiori di numero scrivendo il risultato.
51. Progettare un algoritmo che, dato un valore numerico numero, legga n valori e conti quanti sono i valori maggiori di numero, quanti i valori uguali a numero e quanti quelli minori.
52. Progettare un algoritmo che, dato un valore numerico numero, legga n numeri e conti quanti di questi sono multipli di numero scrivendo il risultato.
53. Progettare un algoritmo che legga una sequenza di valori numerici fino a che la loro somma è minore di 100 e scriva la somma ottenuta e quanti sono i valori letti.
54. Progettare un algoritmo che, dati due valori numerici numMinore e numMaggiore, legga n valori e conti quanti di essi sono compresi tra numMinore e numMaggiore scrivendo il risultato.
55. Progettare un algoritmo che permetta di inserire un numero compreso nell'intervallo [1-1000] e, partendo da questo, sommi i primi cinque numeri che sono divisibili sia per 3, sia per 5, visualizzando sia i numeri che rispettano il criterio, sia il risultato finale.
56. Progettare un algoritmo che prenda in input un valore numerico maggiore o uguale a 10 e restituisca la somma di tutte le cifre che lo compongono.
57. Progettare un algoritmo che indichi a video se un numero è o meno palindromo.
Un numero palindromo è un numero che invertito restituisce lo stesso numero originale, ad esempio:
 - -1221 è palindromo;
 - 707 è palindromo;
 - 11212 non è palindromo;
 - 121 è palindromo;
 - -12321 è palindromo;
 - 1001 è palindromo;
 - -744117 non è palindromo.

Per verificare se un numero è palindromo potrebbe essere utile determinare l'inverso del numero dato, da confrontare con il numero originale.

Per trovare l'inverso di un numero ricordati che puoi estrarre una cifra dopo l'altra con il % e spostarla di posizione moltiplicandola per 10.

58. Progettare un algoritmo che, dato in input un numero intero positivo, visualizzi la somma della cifra più significativa e della cifra meno significativa del numero. Ad esempio:

- Input: 252 → Output: 4
 - Input: 257 → Output: 9
 - Input: 0 → Output: 0
 - Input: 5 → Output: 10
 - Input: -10 → Output: ERRORE
59. Progettare un algoritmo che dato in input un numero intero positivo visualizzi la somma delle sue cifre pari.
60. Progettare un algoritmo che prenda in input due numeri nell'intervallo [10-99], e se hanno una cifra in comune visualizzi la stringa "I due numeri condividono cifre", altrimenti visualizzi "I due numeri non condividono cifre".
61. Progettare un algoritmo che, dato in input un numero intero positivo espresso in base 10, fornisca la sua conversione in base 2 (non si utilizzino gli array). Per visualizzare correttamente il numero binario si utilizzi la tecnica per determinare l'inverso di un numero consigliata nell'esercizio per individuare se il numero è palindromo.
62. Progettare un algoritmo che prenda in input un numero intero maggiore di 1 e visualizzi tutti i suoi fattori primi. Ad esempio, i fattori primi di 6 sono 1, 2, 3, 6.
63. Progettare un algoritmo che preso in input un numero intero positivo indichi se è o meno un numero perfetto. Un numero è perfetto se è uguale alla somma dei suoi divisori propri, ad esempio, 6 è un numero perfetto in quanto è somma dei suoi divisori propri: $1 + 2 + 3 = 6$.
64. Progettare un algoritmo che, leggendo n valori numerici, verifichi se essi sono forniti in ordine crescente o meno.
65. Progettare un algoritmo che, dato un valore numerico k, legga n coppie di valori e conti quante di queste coppie hanno come prodotto il valore k.
66. Dati n valori numerici in ordine crescente, progettare un algoritmo che scriva se i numeri forniti a partire dal secondo differiscono ognuno dal precedente di un valore costante. In caso affermativo l'algoritmo deve scrivere il valore della differenza, in caso negativo l'algoritmo deve scrivere il valore massimo delle differenze.
67. I seguenti problemi sono di difficoltà incrementale, iniziate dal più semplice e, una volta risolto, provate la versione successiva.
- a. [EASY] Progettare un programma in grado di visualizzare il totale di un ordine. Per ogni merce dell'ordine si dovrà inserire la quantità e il prezzo unitario. Se il totale dell'ordine supera i 200€ verrà applicato uno sconto del 15%.
 - b. [NORMAL] Progettare un programma in grado di visualizzare il totale di un ordine. Per ogni merce dell'ordine si dovrà inserire la quantità e il prezzo unitario. Se il totale dell'ordine supera i 200€ verrà applicato uno sconto del 15%, mentre se si superano i 500€ sarà applicato uno sconto del 30%.
 - c. [HARD] Progettare un programma in grado di visualizzare il totale di un ordine. Per ogni merce dell'ordine si dovrà inserire la quantità e il prezzo unitario. Se il totale dell'ordine supera i 200€ verrà applicato uno sconto del 15%, mentre se si

superano i 500€ sarà applicato uno sconto del 30%. Inoltre, se viene richiesta la consegna a domicilio si dovrà pagare 5€ in più.

68. Dato un valore numerico costante k (non necessariamente intero), l' N -esimo numero di Bernoulli è dato dalla somma dei primi N numeri interi elevati alla potenza k ; per esempio per $N = 5$:
- $$1^k + 2^k + 3^k + 4^k + 5^k$$
- Progettare un algoritmo che determini, a partire dai valori della costante k e del numero N , il numero di Bernoulli relativo.
69. La media geometrica di N numeri $x_1, x_2, \dots, x_N$ è data dalla seguente formula:
- $$(x_1 * x_2 * \dots * x_N)^{1/N}$$
- Progettare un algoritmo che calcoli la media geometrica di N numeri positivi inseriti dall'utente.
70. La società di assicurazioni "Io speriamo che me la cavo" ha stipulato N contratti di assicurazione su motociclette secondo la seguente formula:
- se la moto ha cilindrata maggiore di 350, il costo dell'assicurazione è uguale a una quota fissa Q più € 30 per ogni mille euro del prezzo della moto;
 - altrimenti il costo dell'assicurazione è uguale alla quota fissa Q più € 20 per ogni mille euro del prezzo della moto.
- Progettare un algoritmo in grado di fornire in output quanto ha incassato l'assicurazione.
71. Lo stipendio di un dipendente dell'azienda "Ma quanto mi costi" è formato da 3 parti: salario accessorio (A), stipendio base (B), compensi aggiuntivi (C). Sulla parte A si applica la trattenuta del 19%, sulla parte B il 16%, sulla parte C il 2%. Sul totale $A+B+C$ viene trattenuto un ulteriore 0.5%.
- Progettare un algoritmo che per ognuno dei dipendenti visualizzi il numero di matricola, il totale delle trattenute e lo stipendio finale netto. Si visualizzi infine il totale degli stipendi lordi e il totale delle trattenute di tutti i dipendenti.
72. Progettare un algoritmo che determini in un gruppo di persone quante risultano essere sottopeso, quante normopeso, quante in sovrappeso e quante obese.
- Si consideri che l'Indice di Massa Corporea (IMC) di una persona si determina con la seguente formula:
- $$IMC = \text{peso (kg)} / \text{altezza}^2 (\text{m}^2)$$
- e che l'Organizzazione Mondiale della Sanità individua le seguenti quattro categorie:
- sottopeso: IMC inferiore a 19;
 - normopeso: IMC nell'intervallo [19, 24];
 - sovrappeso: IMC nell'intervallo (24, 30];
 - obeso: IMC superiore a 30.

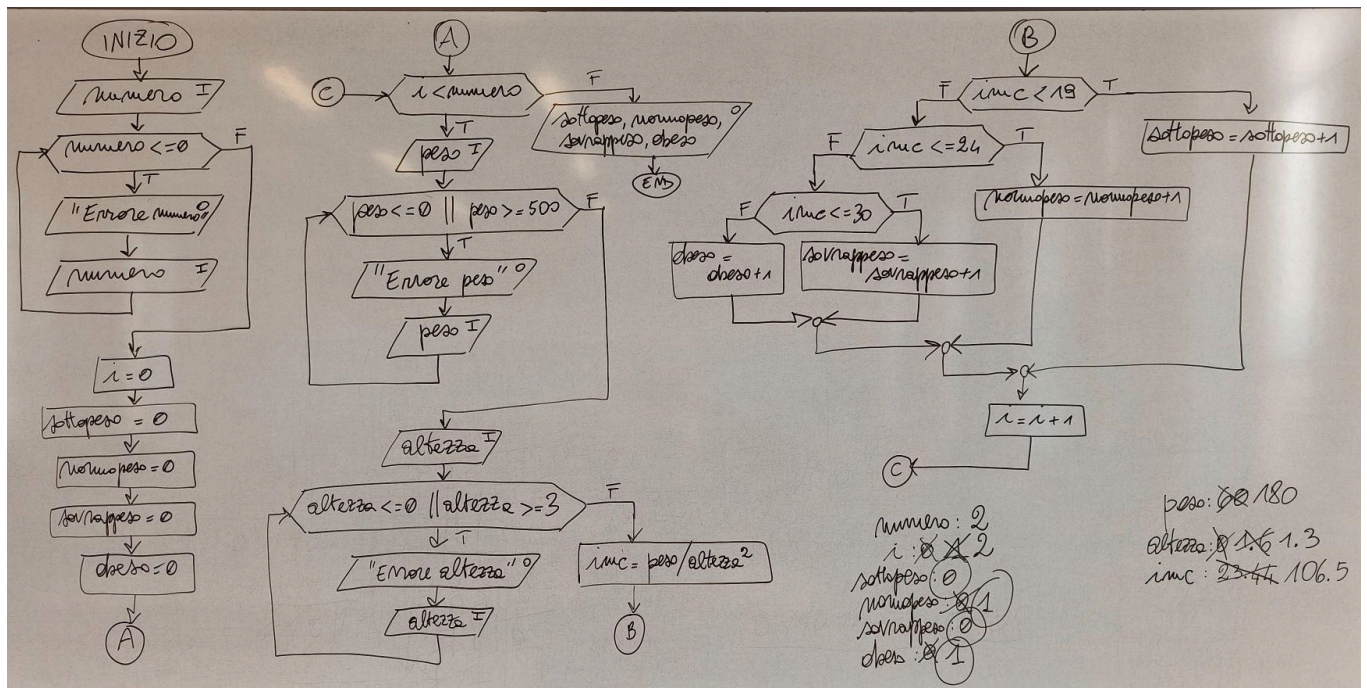
Obiettivo

Effettuato un calcolo di interesse, viene fornito il conteggio dell'appartenenza del calcolo ad uno degli intervalli forniti.

Tabella delle variabili

Nome	Tipo	Descrizione	I/O/L	Dominio validità
numero	int	Numero di persone analizzate	I	numero > 0
i	int	Contatore di ciclo per contare le	L	-

		persone analizzate		
peso	float	Peso di ogni persona analizzata	I	$\text{peso} > 0 \ \&\& \ \text{peso} < 500$
altezza	float	Altezza di ogni persona analizzata	I	$\text{altezza} > 0 \ \&\& \ \text{altezza} < 3$
imc	float	Indice di massa corporea calcolato per ogni persona	L	-
sottopeso	int	Contatore delle persone sottopeso	O	$\text{sottopeso} \geq 0$
normopeso	int	Contatore delle persone normopeso	O	$\text{normopeso} \geq 0$
sovrappeso	int	Contatore delle persone sovrappeso	O	$\text{sovrappeso} \geq 0$
obeso	int	Contatore delle persone obese	O	$\text{obeso} \geq 0$



73. Relativamente al progetto dell'IMC si progetti un algoritmo che, dopo aver calcolato il numero di persone di un gruppo che risultano sottopeso (s), normopeso (n), sovrappeso (S) e obeso (o), visualizzi tali valori con un istogramma orizzontale utilizzando il carattere '*'.

Esempio: sottopeso = 3, normopeso = 5, sovrappeso = 8, obeso = 2

```
s: ***
n: *****
S: *****
o: **
```

74. Relativamente al progetto dell'IMC si progetti un algoritmo che, dopo aver calcolato il numero di persone di un gruppo che risultano sottopeso, normopeso, sovrappeso e

obese, visualizzi tali valori con un istogramma verticale al contrario utilizzando il carattere '*'.

Esempio: sottopeso = 3, normopeso = 5, sovrappeso = 8, obese = 2

```
s n S o
* * * *
* * * *
* * *
* * *
* *
* *
* *
*
*
*
```

75. Relativamente al progetto dell'IMC si progetti un algoritmo che, dopo aver calcolato il numero di persone di un gruppo che risultano sottopeso, normopeso, sovrappeso e obese, visualizzi tali valori con un istogramma verticale utilizzando il carattere '*'.

Esempio: sottopeso = 3, normopeso = 5, sovrappeso = 8, obese = 2

```
*
*
*
* *
* *
* * *
* * * *
* * * *
s n S o
```

76. Progettare un algoritmo che preveda l'inserimento delle altezze in centimetri di un gruppo di persone, e visualizzi l'altezza minima e il numero di persone che hanno un'altezza pari a quella minima. Si effettuino tutti i controlli necessari sui valori inseriti in input.
77. In una gara podistica si vogliono visualizzare il tempo migliore, il tempo peggiore e il tempo medio di percorrenza. Si preveda l'inserimento dei tempi di percorrenza dei diversi atleti effettuando tutti i controlli necessari sui valori inseriti in input. I tempi di percorrenza vengono forniti in secondi.
78. Progettare un algoritmo che permetta di assemblare uno scatolone di pacchi di farina usando pacchi da 5kg e 1kg. L'algoritmo deve prevedere l'inserimento dei seguenti dati:

- numero totale di kg da inserire in uno scatolone;
- numero di pacchi da 5kg disponibili;
- numero di pacchi da 1kg disponibili.

Uno scatolone può ritenersi assemblato se sono presenti almeno tutti i kg che può contenere, considerando la somma dei kg dei pacchi da 5kg e da 1kg. Non è però possibile dividere i pacchi di farina, quindi se lo scatolone può contenere 9kg di farina, e si hanno solo 2 pacchi da 5kg e nessuno da 1kg, non è possibile assemblare lo scatolone perché i pacchi da 5kg non possono essere divisi. Se invece si hanno a disposizione 1 pacco da 5kg e 5 pacchi da 1kg è possibile assemblare lo scatolone da 9kg perché avanza un pacco da 1kg intero e questo è considerato ammissibile.

Si visualizzi come output una scritta che indichi se lo scatolone è o meno stato assemblato.

79. Progettare un algoritmo che permetta di determinare il più grande numero primo fattore di un numero intero positivo dato. Si ricordi che per il [teorema fondamentale dell'aritmetica](#) un qualsiasi numero naturale maggiore di 1 o è un numero primo o si può esprimere come prodotto di numeri primi. Nel caso in cui il numero inserito sia minore o uguale ad 1 si richieda l'inserimento del numero, in quanto 0 e 1 non sono considerati primi.

Esempi:

Input: 21 → Output: 7 perché 7 è il più grande numero primo ($3 * 7 = 21$)

Input: 217 → Output: 31 perché 31 è il più grande numero primo ($7 * 31 = 217$)

Input: 13 → Output: 13 poiché 13 è il più grande numero primo ($13 = 13$)

Input: 0 → Richiedere l'immissione dell'input

Input: 45 → Output: 5 perché 5 è il più grande numero primo ($3 * 3 * 5 = 45$)

Input: -1 → Richiedere l'immissione dell'input

80. Progettare un algoritmo che prende in input un numero maggiore o uguale a 5 e visualizzi un quadrato con le due diagonali, con una base e un'altezza pari al numero inserito in input.

Ad esempio:

Input: 5 → Output:

** **

* * *

** **

Input: 8 → Output:

** **

* * * *

* ** *

* * * *

** **

81. Un grande magazzino ha 4 reparti identificati dai numeri 1, 2, 3 e 4. La direzione decide di applicare degli sconti ai prodotti dei diversi magazzini, differenziandoli per magazzino. Si progetti un algoritmo in grado di richiedere in input le quattro percentuali di sconto da applicare ai prodotti dei vari reparti e, successivamente richieda in input N prodotto con il relativo reparto di appartenenza e il prezzo, e visualizzi per ciascun prodotto inserito il prezzo scontato.

82. Progettare un algoritmo in grado di svolgere le quattro operazioni fondamentali su due numeri.

Visualizzare inizialmente il seguente menù:

1) Somma

2) Sottrai

3) Moltiplica

4) Dividi

5) Esci

Se viene scelta una delle opzioni da 1 a 4 viene svolta l'operazione aritmetica richiesta sui due numeri visualizzando il risultato. Successivamente viene nuovamente visualizzato il menù e richiesto l'inserimento di due numeri.

Se invece viene scelta l'opzione 5 il programma deve terminare, salutando cortesemente l'utente. Usare lo switch-case per il menù.

83. Progettare un algoritmo che preveda l'inserimento delle età degli studenti della tua classe e fornisca il numero di occorrenze del più grande.

84. Progettare un algoritmo che visualizzi la rendita annuale di un investimento effettuato presso la banca "Più soldi per tutti".

La banca visualizza il piano di investimento usando il capitale iniziale in euro, la percentuale di interesse e il numero di anni dell'investimento.

Il calcolo degli interessi si effettua tramite la seguente formula:

$\text{interessi} = \text{capitale} * \text{tasso} / 100$

e questi verranno sommati di anno in anno al capitale.

Ad esempio se la somma dell'investimento è di 30000 euro al tasso del 12.5% per 10 anni, il piano annuale dell'investimento dovrà essere il seguente:

Anno	Interesse	Capitale
1	3750.00	33750.00
2	4218.75	37968.75
3	4746.09	42714.84
4	5339.35	48054.19
5	6006.77	54060.96
6	6757.62	60818.58
7	7602.32	68420.90
8	8552.61	76973.51
9	9621.68	86595.19
10	10824.39	97419.58

85. Progettare un algoritmo che converta un numero binario in un numero in base 10. Il numero binario è rappresentato su N bit, e il valore di N dovrà essere fornito dall'utente. L'utente dovrà inserire le cifre del numero binario un bit alla volta, partendo dal bit meno significativo (ossia dal bit di peso 2^0). Il programma visualizzerà il numero in base 10 corrispondente (non si utilizzino gli array).

86. Progettare un algoritmo che calcoli il valore massimo e minimo di un insieme di N numeri inseriti da tastiera.

87. Progettare un algoritmo che analizzi una sequenza di numeri. I numeri dovranno essere inseriti da tastiera e l'algoritmo dovrà visualizzare i seguenti risultati:

- quanti sono i numeri positivi, nulli e negativi;
- quanti sono i numeri pari e dispari;
- se la sequenza dei numeri inseriti è crescente, decrescente oppure ne crescente, nè decrescente.

Si osservi che una sequenza è crescente se ogni numero è maggiore del precedente, è decrescente se ogni numero è minore del precedente, mentre non è crescente e neanche decrescente in tutti gli altri casi.

88. Progettare un algoritmo che calcoli il massimo comune divisore (MCD) di due numeri interi positivi. Il MCD è definito come il massimo tra i divisori comuni ai due numeri. Dati due numeri, n_1 e n_2 , il MCD di n_1 e n_2 è il massimo tra i numeri che sono divisori (con resto uguale a zero) sia di n_1 che di n_2 . In particolare si supponga che n_1 sia minore di n_2 , il MCD è il massimo tra i numeri compresi tra 1 e n_1 che sono divisori (con resto uguale a zero) sia di n_1 che di n_2 .
89. Progettare un algoritmo che calcoli il minimo comune multiplo (MCM) di due numeri interi positivi. Dati due numeri interi n_1 e n_2 , il minimo comune multiplo è il più piccolo numero m che è divisibile (con resto pari a zero) sia per n_1 che per n_2 .
90. Progettare un algoritmo che chieda in input un numero maggiore di 1 e visualizzi un quadrato di asterischi (*) di lato pari al valore numerico inserito.
Ad esempio:
Input: 4 → Output:

91. Progettare un algoritmo che chieda in input un numero maggiore di 1 e visualizzi un triangolo rettangolo di asterischi (*) di altezza e base pari al valore numerico inserito.
Ad esempio:
Input: 4 → Output:
*
**

92. Progettare un algoritmo che chieda in input un numero maggiore di 1 e visualizzi un triangolo rettangolo di asterischi (*) di altezza e base pari al valore numerico inserito, con il lato verticale spostato verso destra.
Ad esempio:
Input: 4 → Output:
 *
 **

93. Progettare un algoritmo che chieda in input un numero maggiore di 1 e visualizzi un quadrato i cui lati siano formati da asterischi (*) di lato pari al valore numerico inserito.
Ad esempio:
Input: 5 → Output:

* *
* *
* *

94. Progettare un algoritmo che chieda in input un numero maggiore di 1 e visualizzi un triangolo rettangolo i cui lati siano formati da asterischi (`\*`) di altezza e base pari al valore numerico inserito.

Ad esempio:

Input: 5 → Output:

```
*  
**  
* *  
*  *  
*****
```

95. bla bla

Programmazione modulare²

Funzioni

C Tutorial

4. C Programming Arrays by [Programiz](#)
 - a. [C Functions](#)
 - b. [C User-defined functions](#)
 - c. [Types of User-defined Functions in C Programming](#)
 - d. [C Recursion](#)
 - e. [C Storage Class](#)
 - f. [C Pointers](#)
 - g. [C Pass Addresses and Pointers](#)
 - h. [C Preprocessor and Macros](#)
5. VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)
 - a. [Function Basics](#)
 - b. [Function Parameters VS. Arguments](#)
 - c. [Recursion](#)
 - d. [Pass By Reference](#)
 - e. [Local vs. Global Variables](#)
 - f. [Variable Scope](#)
 - g. **[Introduction to Pointers](#)** ← **IMPORTANTE!!!** *Per ora puoi fermarti dopo la spiegazione della funzione swap(), al minuto 18:23, la funzione malloc() la vedremo più avanti*
 - h. [int *p vs int* p Pointer Declarations](#)
 - i. [How To Create A Library And Split A Program Across Multiple Files](#)
 - j. [lvalues and rvalues](#)
 - k. [int main\(\) vs int main\(void\)](#)
 - l. [Static Local Variables](#) *usa i file per fare l'esempio così inizi a fare pratica*
 - m. [Static Global Variables](#)
 - n. [Register Variables](#)

² "Modular code and how to structure an embedded C project." 20 ott. 2019, <https://www.microforum.cc/blogs/entry/46-modular-code-and-how-to-structure-an-embedded-c-project/>.

Passaggio dei parametri per valore

Somma di due numeri

Nel seguente esempio viene mostrato l'uso di una funzione per effettuare la somma di due numeri interi passati come parametri. All'interno della funzione `main()` viene invocata la funzione

```
risultato = somma(operando1, operando2)
```

che riceve come **parametri attuali** le variabili **operando1** e **operando2** locali al `main()` e, restituisce un valore che viene salvato nella variabile **risultato** anch'essa locale al `main()`. I **parametri attuali** di una funzione sono i valori reali sui quali la funzione lavora. I **parametri attuali** possono essere valori effettivi o variabili, ma in ogni caso il loro valore viene copiato all'interno dei **parametri formali** della funzione. Il **valore restituito dalla funzione** può essere salvato all'interno di una variabile.

Nella libreria `operazioni.h` è stata dichiarata la funzione

```
int somma(int firstOperand, int secondOperand);
```

mentre la sua definizione è stata fatta nel file `operazioni.c`

```
int somma(int firstOperand, int secondOperand) {  
    int sum;  
  
    sum = firstOperand + secondOperand;  
    return sum;  
}
```

La funzione ha due **parametri formali** `firstOperand` e `secondOperand` di tipo `int` e restituisce un valore intero alla funzione `main()` che la invoca, cosa visibile dal tipo `int` posto prima del nome della funzione `somma` e dall'istruzione `return sum;` che restituisce al chiamante (`main()`) il risultato della somma. I **parametri formali** sono delle variabili locali alla funzione, che sono dedicate al ricevimento del contenuto passato dai **parametri attuali**. All'interno del corpo della funzione vengono sempre usati i **parametri formali** che, nel caso di **passaggio dei parametri per valore**, **contengono la copia del valore dei parametri attuali**.

La funzione `somma()` è stata dichiarata all'interno della libreria `operazioni.h` che viene inclusa all'inizio del file `main.c`, mentre la sua definizione è stata fatta nel file `operazioni.c`. Questo permette di dichiarare la funzione `somma()` prima del suo utilizzo consentendone l'uso nella funzione `main()`. Spostando la definizione della funzione in un file separato (`operazioni.c`) si rende anche possibile una organizzazione modulare del codice **incapsulando** le funzioni. L'incapsulamento ci consente di confinare in un unico file (`operazioni.c`) le future espansioni e modifiche delle operazioni aritmetiche. Si veda il paragrafo [File header definiti dall'utente](#) per maggiori dettagli sulla definizione di un proprio file `.h`.

main.c

```
/**
 * Dati due interi calcolare la somma
 * @author mgm
 */
#include <stdio.h>
#include "lib/operazioni.h"

int main(void) {
    int operando1;
    int operando2;
    int risultato;

    // Inserimento input
    printf("Sono in main - Inserire il 1° operando: ");
    scanf("%d", &operando1);
    printf("Sono in main - Inserire il 2° operando: ");
    scanf("%d", &operando2);

    // Calcolo della somma
    risultato = somma(operando1, operando2);

    // Visualizzazione del risultato
    printf("Sono in main - %d + %d = %d\n", operando1, operando2, risultato);
    return 0;
}
```

operazioni.h

```
#ifndef SOMMA_H
#define SOMMA_H
int somma(int firstOperand, int secondOperand);
#endif
```

operazioni.c

```
/**
 * Somma due valori interi
 * @param firstOperand Primo operando
 * @param secondOperand Secondo operando
 */
int somma(int firstOperand, int secondOperand) {
    int sum;

    sum = firstOperand + secondOperand;
    return sum;
}
```

Una esecuzione dell'applicazione genera i seguenti output a fronte degli input forniti.

```
Sono in main - Inserire il 1° operando: 5
Sono in main - Inserire il 2° operando: 8
```

Sono in main - $5 + 8 = 13$

Come si può vedere, a fronte degli input 5 e 8 effettuati all'interno della funzione `main()`, la funzione `somma()` effettua la somma dei due valori e restituisce il risultato al `main()` che lo salva nella sua variabile locale **risultato**.

Questo esempio mostra che l'invocazione di una funzione che effettua il **passaggio dei parametri attuali per valore serve** nel momento in cui **non sono necessarie modifiche ai parametri attuali stessi**. Infatti, in questo caso è necessario mantenere inalterati i valori contenuti nelle variabili locali al `main()` **operando1** e **operando2** in quanto sono necessari per la visualizzazione finale del risultato.

Scambio di due numeri

Nel seguente esempio viene mostrato l'uso **errato** di una funzione per effettuare lo scambio di due numeri passati come parametri. All'interno della funzione `main()` viene invocata la funzione

```
scambio(primoNum, secondoNum)
```

che riceve come **parametri attuali** le variabili `primoNum` e `secondoNum` locali al `main()`. I **parametri attuali** di una funzione sono i valori reali sui quali la funzione lavora. I **parametri attuali** possono essere valori effettivi o variabili, ma in ogni caso il loro valore viene copiato all'interno dei **parametri formali** della funzione.

Nella libreria `utility.h` è stata dichiarata la funzione

```
void scambio(int first, int second);
```

mentre la sua definizione è stata fatta nel file `utility.c`

```
void scambio(int first, int second) {  
    int temp;  
  
    // Scambio i valori  
    temp = first;  
    first = second;  
    second = temp;  
  
    printf("Sono in scambio - 1° numero: %d\n", first);  
    printf("Sono in scambio - 2° numero: %d\n", second);  
}
```

La funzione ha due **parametri formali** `first` e `second` di tipo `int` e non restituisce alcun valore alla funzione `main()` che la invoca, cosa visibile dal tipo `void` posto prima del nome della funzione `scambio`. I **parametri formali** sono delle variabili locali alla funzione, che sono dedicate al ricevimento del contenuto passato dai **parametri attuali**. All'interno del corpo della funzione vengono sempre usati i **parametri formali** che, nel caso di **passaggio dei parametri per valore**, contengono la copia del valore dei **parametri attuali**.

La funzione `scambio()` è stata dichiarata all'interno della libreria `utility.h` che viene inclusa all'inizio del file `main.c`. Questo permette di dichiarare la funzione `scambio()` prima del suo utilizzo consentendone l'uso nella funzione `main()`.

La funzione `scambio()` è stata dichiarata all'interno della libreria `utility.h` che viene inclusa all'inizio del file `main.c`, mentre la sua definizione è stata fatta nel file `utility.c`. Questo permette di dichiarare la funzione `scambio()` prima del suo utilizzo consentendone l'uso nella funzione `main()`. Spostando la definizione della funzione in un file separato (`utility.c`) si rende anche possibile una organizzazione modulare del codice **incapsulando** le funzioni. L'**incapsulamento** ci consente di confinare in un unico file (`utility.c`) le future espansioni e modifiche delle operazioni di scambio. Si veda il paragrafo [File header definiti dall'utente](#) per maggiori dettagli sulla definizione di un proprio file `.h`.

main.c

```
/**
 * Dati due interi li scambia di posizione
 * @author mgm
 */
#include <stdlib.h>
#include <stdio.h>
#include "lib/utility.h"

int main() {
    int primoNum;
    int secondoNum;

    // Inserimento input
    printf("Sono in main - Inserire il 1° numero: ");
    scanf("%d", &primoNum);
    printf("Sono in main - Inserire il 2° numero: ");
    scanf("%d", &secondoNum);

    // Scambio i numeri
    scambio(primoNum, secondoNum);

    // Visualizzo i numeri scambiati
    printf("Sono in main - 1° numero: %d\n", primoNum);
    printf("Sono in main - 2° numero: %d\n", secondoNum);
    return 0;
}
```

utility.h

```
#ifndef UTILITY_H
#define UTILITY_H

void scambio(int first, int second);

#endif
```

utility.c

```
#include <stdio.h>

/**
 * Scambia due valori interi
 * @param first Primo valore intero
 * @param second Secondo valore intero
 */
void scambio(int first, int second) {
    int temp;

    // Scambio i valori
    temp = first;
    first = second;
```

```
second = temp;

printf("Sono in scambio - 1° numero: %d\n", first);
printf("Sono in scambio - 2° numero: %d\n", second);
}
```

Una esecuzione dell'applicazione genera i seguenti output a fronte degli input forniti.

```
Sono in main - Inserire il 1° numero: 4
Sono in main - Inserire il 2° numero: 9
Sono in scambio - 1° numero: 9
Sono in scambio - 2° numero: 4
Sono in main - 1° numero: 4
Sono in main - 2° numero: 9
```

Come si può vedere, a fronte degli input 4 e 9 effettuati all'interno della funzione `main()`, la funzione `scambio()` effettua lo spostamento dei due valori al suo interno, come ci si aspetterebbe, ma una volta tornati nella funzione `main()` la visualizzazione del contenuto delle variabili originali **primoNum** e **secondoNum** dimostra che lo scambio non è avvenuto.

Affinché lo scambio dei valori risulti permanente anche nella funzione `main()` è necessario cambiare la modalità di invocazione della funzione, prevedendo il **passaggio dei parametri attuali per indirizzo**, che prevede la **modifica del contenuto dei parametri attuali**.

Questo esempio mostra che l'invocazione di una funzione che effettua il **passaggio dei parametri attuali per valore serve** nel momento in cui **non sono necessarie modifiche ai parametri attuali stessi**. Se invece fosse **necessario** ottenere una **modifica del contenuto dei parametri attuali**, allora è necessario cambiare la modalità di invocazione della funzione, prevedendo il **passaggio dei parametri attuali per indirizzo**.

Passaggio dei parametri per indirizzo

Scambio di due numeri

Nel seguente esempio viene mostrato l'uso di una funzione per effettuare lo scambio di due numeri passati come parametri. All'interno della funzione `main()` viene invocata la funzione

```
scambio(&primoNum, &secondoNum)
```

che riceve come **parametri attuali** gli **indirizzi di memoria** (& - operatore di **referenziamento**) delle variabili `primoNum` e `secondoNum` locali al `main()`. I **parametri attuali** di una funzione sono i valori reali sui quali la funzione lavora e, in questo caso, corrispondono agli **indirizzi di memoria** delle due variabili e non al loro contenuto. Questi **parametri attuali** vengono copiati all'interno dei **parametri formali** della funzione che, in questo caso, si aspetta di ricevere due **indirizzi di memoria**.

Nella libreria `utility.h` è stata dichiarata la funzione

```
void scambio(int *first, int *second)
```

che ha due **parametri formali** `first` e `second` di tipo `int*`, cioè sono di **tipo puntatore ad intero** (* - **variabile puntatore al tipo indicato**). La funzione non restituisce alcun valore alla funzione `main()` che la invoca, cosa visibile dal tipo `void` posto prima del nome della funzione `scambio`. I due **parametri formali** di tipo `int*` indicano che i due **parametri attuali** passati dalla funzione `main()` devono essere gli **indirizzi di memoria** di due variabili di tipo `int` locali al `main()`. All'interno del corpo della funzione vengono sempre usati i **parametri formali** che, nel caso di **passaggio dei parametri per indirizzo**, **contengono l'indirizzo di memoria dei parametri attuali**.

La funzione `scambio()` è stata dichiarata all'interno della libreria `utility.h` che viene inclusa all'inizio del file `main.c`. Questo permette di dichiarare la funzione `scambio()` prima del suo utilizzo consentendone l'uso nella funzione `main()`. Si veda il paragrafo [File header definiti dall'utente](#) per maggiori dettagli sulla definizione di un proprio file `.h`.

All'interno della funzione `scambio()`, per accedere ai valori effettivi da scambiare e non usare gli indirizzi di memoria presenti nei **parametri formali**, è necessario usare l'**operatore di dereferenziazione** `*`. Nel corpo di una funzione questo operatore, posto prima del nome di un **parametro formale**, permette di raggiungere il valore memorizzato nella variabile il cui indirizzo di memoria è memorizzato nel **parametro formale**.

Quindi l'operatore `*` assume due significati:

- quando viene **usato in una dichiarazione crea una variabile puntatore**, ad esempio: `void scambio(int* first, int* second)`
- quando **non è usato in una dichiarazione diventa l'operatore di dereferenziazione**, cioè permette di recuperare il valore memorizzato all'indirizzo di memoria contenuto nella variabile puntatore, ad esempio: `temp = *first;`

main.c

```
/**
 * Dati due interi li scambia di posizione
 * @author mgm
 */
#include <stdlib.h>
#include <stdio.h>
#include "lib/utility.h"

int main() {
    int primoNum;
    int secondoNum;

    // Inserimento input
    printf("Sono in main - Inserire il 1° numero: ");
    scanf("%d", &primoNum);
    printf("Sono in main - Inserire il 2° numero: ");
    scanf("%d", &secondoNum);

    // Quali sono gli indirizzi di memoria dei parametri attuali?
    printf("Sono in main - &primoNum: %p\n", &primoNum);
    printf("Sono in main - &secondoNum: %p\n", &secondoNum);

    // Scambio i numeri
    scambio(&primoNum, &secondoNum);

    // Visualizzo i numeri scambiati
    printf("Sono in main - 1° numero: %d\n", primoNum);
    printf("Sono in main - 2° numero: %d\n", secondoNum);
    return 0;
}
```

utility.h

```
#ifndef UTILITY_H
#define UTILITY_H

void scambio(int *first, int *second);

#endif
```

utility.c

```
#include <stdio.h>

/**
 * Scambia due valori interi
 * @param first Primo valore intero
 * @param second Secondo valore intero
 */
void scambio(int *first, int *second) {
    int temp;
```

```
// Scambio i valori
temp = *first;
*first = *second;
*second = temp;

// Contenuto dei parametri formali
printf("Sono in scambio - contenuto first: %p\n", first);
printf("Sono in scambio - contenuto second: %p\n", second);

printf("Sono in scambio - 1° numero: %d\n", *first);
printf("Sono in scambio - 2° numero: %d\n", *second);
}
```

Una esecuzione dell'applicazione genera i seguenti output a fronte degli input forniti.

```
Sono in main - Inserire il 1° numero: 4
Sono in main - Inserire il 2° numero: 8
Sono in main - &primoNum: 0x7fff83c10294
Sono in main - &secondoNum: 0x7fff83c10290
Sono in scambio - contenuto first: 0x7fff83c10294
Sono in scambio - contenuto second: 0x7fff83c10290
Sono in scambio - 1° numero: 8
Sono in scambio - 2° numero: 4
Sono in main - 1° numero: 8
Sono in main - 2° numero: 4
```

Come si può vedere, a fronte degli input 4 e 9 effettuati all'interno della funzione `main()`, la funzione `scambio()` effettua lo spostamento dei due valori al suo interno, come ci si aspetterebbe e, dato che il **passaggio dei parametri è stato effettuato per indirizzo**, lo scambio risulta anche nella funzione `main()`.

Questo esempio mostra che l'invocazione di una funzione che effettua il **passaggio dei parametri attuali per indirizzo serve** nel momento in cui **sono necessarie modifiche ai parametri attuali stessi**. Se invece **non fosse necessario** ottenere una **modifica del contenuto dei parametri attuali**, allora è necessario cambiare la modalità di invocazione della funzione, prevedendo il **passaggio dei parametri attuali per valore**.

File header definiti dal programmatore

Per gestire in modo modulare il proprio codice è necessario creare un proprio file header assegnandogli un nome significativo e terminandolo con l'estensione **.h**.

Negli esempi mostrati precedentemente sono stati creati:

- il file **operazioni.h** che contiene la dichiarazione della funzione
`int somma(int firstOperand, int secondOperand)`
- Il file **utility.h** che contiene la dichiarazione della funzione
`void scambio(int *first, int *second)`

In entrambi i casi le funzioni sono state usate nella corrispondente funzione `main()`, includendo il file header in cima al `main.c` attraverso la direttiva **#include** nel seguente modo:

```
#include "utility.h"
```

Se lo header che definisce una nuova libreria personale fosse salvato in una directory diversa da quella in cui è salvato il file che contiene la funzione `main()`, sarebbe necessario includere il path relativo della directory della libreria oltre al nome della libreria stessa, come mostrato di seguito, dove si suppone che la libreria `utility.h` è stata salvata nella sottodirectory `lib`:

```
#include "lib/utility.h"
```

Header guard

#ifndef

La direttiva **#ifndef** viene usata nel linguaggio C per controllare se una macro è già stata dichiarata tramite la direttiva **#define**, per evitare di dichiararla più volte nello stesso codice. La direttiva **#ifndef** è la contrazione di **IF Not DEFine** e deve sempre essere chiusa dal corrispondente **#endif**.

La sintassi della direttiva **#ifndef** è la seguente:

```
#ifndef [simbolo]
blocco di istruzioni
#endif
```

Se il simbolo non è definito, il preprocessore aggiunge al codice il blocco di istruzioni tra **#ifndef** e **#endif**, diversamente lo esclude. Dopo l'operazione svolta dal preprocessore, il codice sorgente viene passato al compilatore che crea il file oggetto e, successivamente, il file oggetto viene passato al linker che crea il file eseguibile. Per maggiori informazioni sulle fasi di compilazione di un programma C si veda il capitolo [Compilazione e interpretazione dei programmi](#).

Nell'esempio della dichiarazione della funzione di scambio di due numeri lo header guard è stato usato nel seguente modo:

```
#ifndef UTILITY_H
#define UTILITY_H
void scambio(int *first, int *second);
#endif
```

Con la direttiva

```
#define UTILITY_H
```

è stata dichiarata la macro `UTILITY_H`. Alla macro non viene assegnato alcun valore, ma viene comunque definita.

Ponendo questa macro tra gli header guard `#ifndef UTILITY_H` e `#endif`, si indica al preprocessore che se il file `utility.h` viene incluso più volte nella stessa applicazione, la macro `UTILITY_H` deve essere definita una sola volta. Questo implica anche che la dichiarazione della funzione `void scambio(int *first, int *second);` viene fatta una sola volta, l'unica volta che viene definita la macro `UTILITY_H`. In pratica si usa la definizione della macro `UTILITY_H` per garantire l'inclusione univoca di tutte le dichiarazioni che si trovano tra `#ifndef` e `#endif`.

`#ifdef`

Per completezza si rimanda al seguente articolo per la direttiva `#ifdef`:

[C Preprocessor and Macros](#) by [Programiz](#).

#pragma once³

Al posto dell'uso dello header guard **#ifndef** in combinazione con la definizione di una macro, è possibile gestire le dichiarazioni delle funzioni usando anche l'istituzione del precompilatore **#pragma once**.

Questa istruzione del precompilatore non è standard, ma viene supportata dalla maggior parte dei compilatori. Nonostante sia non-standard, viene comunque ampiamente utilizzata nei grandi progetti in C, perché aumenta la leggibilità del codice e velocizza il processo di precompilazione. Infatti con **#pragma once** il precompilatore non deve controllare se un simbolo è già stato definito per ogni **#include** dello stesso header .

Ad esempio questo header guard:

```
#ifndef UTILITY_H
#define UTILITY_H void scambio(int *first, int *second);
#endif
```

può essere sostituito con:

```
#pragma once
void scambio(int *first, int *second);
```

La direttiva **#pragma once** potrebbe creare problemi nel caso lo stesso header fosse presente in posizioni diverse del progetto. Infatti alcuni compilatori potrebbero interpretarli come due header diversi, causando errori in fase di compilazione. Con gli header guard invece questo problema non si pone.

³ Scritto dallo studente Andrea Como, a.s. 2023/2024

C Tutorial

6. VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)
 - a. [Include Guards](#) ← *l'argomento è spiegato usando le struct che vedremo più avanti, ma cerca di comprendere come usare gli include guards, senza preoccuparti della complessità del codice*

Approfondimenti

C Tutorial

Variadic functions: funzioni con un numero variabile di parametri

7. Function with variable-length parameter lists by [GNU Operating System](#)
 - a. [Variable-Length Parameter Lists](#), [GNU C Language Manual](#)
8. VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)
 - a. [How to create functions with a variable number of arguments using stdarg.h](#)

Direttive condizionali del pre processore

9. VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)
 - a. [#ifdef #ifndef Conditional Compilation Directives](#)
 - b. [#if #elif #else Conditional Compilation Directives](#)

Dalla programmazione strutturata alla programmazione modulare

La **[programmazione strutturata](#)**, risalente agli anni '70 del secolo scorso, fu avveniristica in quanto permise di progettare programmi privi della frammentazione dovuta alla presenza di istruzioni GOTO e return, che permettevano salti condizionati o incondizionati in qualsiasi punto del programma ([spaghetti code](#)). Con la programmazione strutturata vengono introdotte le strutture fondamentali di sequenza, selezione e iterazione che permettono di strutturare il codice in modo controllato e più facilmente correggibile.

Le dimensioni di un'applicazione complessa non permettono però di basarsi solo sulla programmazione strutturata, perché richiedono un'organizzazione adeguata di tutti i file che la compongono in modo che sia semplice modificarla senza che si debba cambiare tutta l'applicazione a fronte della modifica di alcune funzionalità. Per raggiungere questa versatilità è necessario organizzare l'applicazione software come un'[architettura](#) ([Software architecture](#)) che segua una progettazione software basata su principi precisi di sviluppo che nel tempo hanno dato origine a molti **[paradigmi di programmazione](#)**.

La **[programmazione modulare](#)** è un paradigma di programmazione del software che enfatizza la separazione delle funzionalità di un programma in **moduli** indipendenti ed intercambiabili, in modo tale che ognuno di essi contenga tutto ciò che è necessario per eseguire un solo aspetto della funzionalità desiderata. Questo paradigma deriva dalla progettazione modulare ([modular design](#)) dei sistemi che spesso viene usata in ambito industriale per la produzione di prodotti costituiti da parti intercambiabili, o nella costruzione di edifici che sono composti da una miriade di componenti sostituibili generalmente in modo indipendente.

In generale la **[programmazione modulare](#)** permette di creare e progettare delle architetture software che riducono la complessità del codice attraverso il principio di [divide et impera](#), evita la duplicazione del codice ri-usandolo in molti progetti e permette di dividere lo sviluppo software in più gruppi di lavoro.

Quando il codice viene diviso in moduli gestibili da singoli gruppi di lavoro, diventa manutenibile in modo più efficiente. I diversi gruppi possono avere proprie aree di specializzazione e nessuno deve avere una comprensione totale dell'intero problema per riuscire a contribuire in modo efficace alla sua soluzione.

Decomposizione funzionale - Functional decomposition

Nella progettazione di architetture software complesse la **decomposizione funzionale** ([functional decomposition](#)) è un processo che suddivide un sistema complesso in diversi sotto-sistemi più piccoli, ognuno con una propria funzione, cioè con una propria **specificità responsabilità**.

La **decomposizione funzionale** è l'applicazione del principio *divide et impera* e viene anche chiamata **separazione delle competenze** ([Separation of concern principle](#) - *SoC principle*), uno dei principi cardini della programmazione modulare, così come di molti altri paradigmi di programmazione. Il **principio SoC** prevede che un'applicazione deve essere sviluppata separando in modo netto le responsabilità nei diversi moduli del sistema, facendo in modo che questi non si sovrappongano e non siano direttamente dipendenti tra di loro, rimanendo delle entità separate, intercambiabili, modificabili e testabili singolarmente. Ogni modulo deve quindi occuparsi di svolgere un solo preciso compito.

Progettare applicazioni che seguono il **principio SoC** prevede quindi la scrittura di moduli che **incapsulano** ([Encapsulation principle](#)) la struttura dei dati e le funzioni che operano su di essi in librerie separate e indipendenti, nascondendo alle altre librerie i dettagli di implementazione. Ogni libreria fornisce una chiara interfaccia di comunicazione, costituita da funzioni, che permettono l'interazione tra i moduli.

Per raggiungere un tale livello di separazione, il codice di ogni modulo deve essere **fortemente coeso** ([Cohesion](#)), cioè gli elementi presenti all'interno di un modulo devono avere una forte relazione tra di loro dovuta al fatto che perseguono tutti un unico scopo e rappresentano un unico concetto. La coesione di un modulo rappresenta in pratica il livello di relazione tra i dati e i metodi contenuti in esso.

Moduli fortemente coesi e che seguono il principio SoC sono indipendenti e facilmente intercambiabili e presentano quindi un **basso accoppiamento** ([Coupling](#)). Questa misura, insieme all'alta coesione, fornisce una misura della bontà del software. Un basso accoppiamento indica che il grado di interdipendenza tra i moduli dell'applicazione è basso e quindi, insieme all'alta coesione, indica che il modulo è facile da leggere, modificare e sostituire.

La scrittura di moduli che seguono tutti i principi esposti riducono la complessità di sviluppo di una applicazione in quanto gran parte della complessità viene nascosta all'interno dei moduli stessi, che vengono sviluppati indipendentemente. I moduli espongono un'interfaccia semplificata ad un livello di astrazione superiore rispetto alla funzione del modulo stesso.

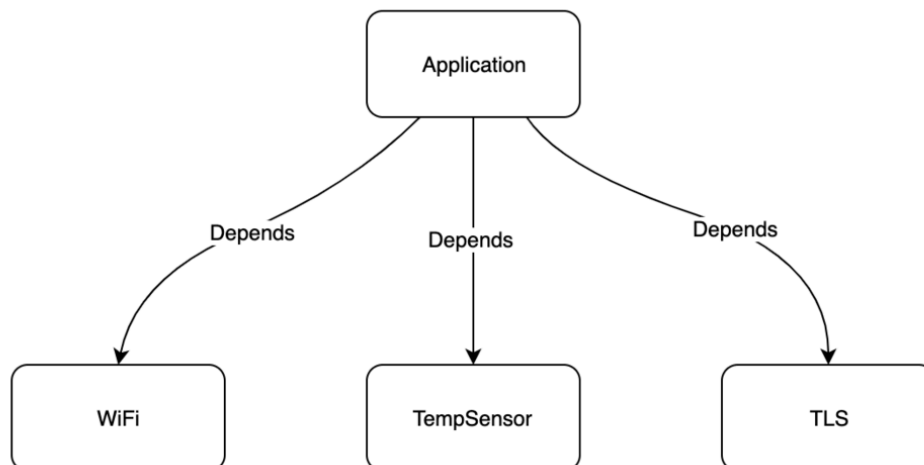
Organizzazione architetturale di un'applicazione⁴

Una delle cose più importanti di un'applicazione è la sua organizzazione architetturale e le modalità per attuarla possono essere diverse, a seconda delle necessità sia aziendali, sia progettuali.

Ciò che si tenta di perseguire con la tecnica presentata di seguito è la creazione di **moduli fortemente coesi** ([cohesion](#)) che **astraggono** ([abstraction](#)) un singolo concetto, **incapsulando** ([encapsulation](#)) tutti i dettagli ed esponendo solo ciò che è strettamente necessario. Queste tecniche permettono quindi di **separare le responsabilità** ([Separation of concern principle](#)) tra diversi **moduli** in modo che risultino **debolmente accoppiati** ([coupling](#)) tra loro. Il modulo deve astrarre un concetto generale ed occuparsene completamente, diventando un componente riutilizzabile in diverse applicazioni.

All'interno di ogni modulo le funzioni che lo compongono devono perseguire gli stessi principi di forte coesione e basso accoppiamento, perseguibili tramite **principio di singola responsabilità** ([Single Responsibility principle](#)). Questo principio indica una precisa tecnica di scrittura delle funzioni in modo che abbiano una sola ragione per cambiare, legata al solo concetto che astraggono.

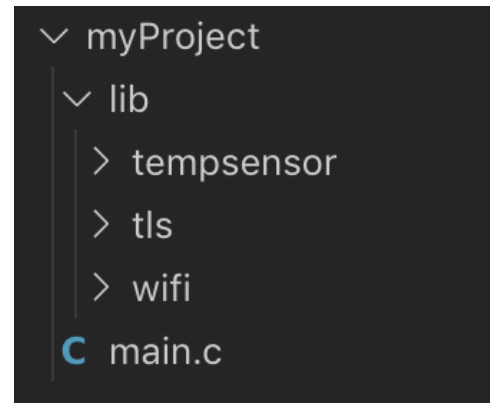
Ad esempio, l'architettura⁵ seguente è quella di un progetto di gestione della temperatura che necessita di tre librerie distinte: una per la gestione dei sensori della temperatura (*TempSensor*), una per la gestione del Wi-Fi ([WiFi](#)) e una per implementare una comunicazione sicura usando il Wi-Fi, che usa il protocollo TLS ([TLS](#)).



⁴ "Modular code and how to structure an embedded C project." 20 ott. 2019, <https://www.microforum.cc/blogs/entry/46-modular-code-and-how-to-structure-an-embedded-c-project/>.

⁵ "Modular code and how to structure an embedded C project." 20 ott. 2019, <https://www.microforum.cc/blogs/entry/46-modular-code-and-how-to-structure-an-embedded-c-project/>.

Per rispettare tutti i principi esposti l'applicazione può organizzare le librerie come mostrato nell'immagine a lato, nella quale il `main.c` ha il solo compito di usare le strutture dati e di invocare le funzioni definite all'interno delle singole librerie poste nella directory `lib`.



Tra le librerie (o moduli) non deve esistere alcuna dipendenza diretta, cioè devono avere un basso accoppiamento. Qualsiasi legame tra le librerie deve figurare come una configurazione che viene impostata (iniettata - *be injected*) direttamente dall'applicazione generale e non direttamente dai singoli moduli. Questo vuol dire che la comunicazione tra i moduli viene gestita nella logica del file `main.c`, senza che i moduli abbiano alcuna conoscenza della cosa. Questo concetto è particolarmente importante perché in questo modo il punto di modifica per iniettare le configurazioni rimane l'applicazione generale e non le singole librerie che così risultano **debolmente accoppiate**.

In linea di principio le librerie non devono essere modificate per essere adattate all'applicazione corrente, bensì deve essere modificata l'applicazione corrente per aderire alle librerie che vuole usare, diversamente le librerie non funzionerebbero correttamente per nessun'altra applicazione che già le sta usando.

Seguendo questa organizzazione architetturale e i principi esposti, l'applicazione rispetta un altro importante principio della programmazione, il [principio Open-Closed](#) che afferma proprio che un modulo deve essere **aperto** (*Open*) **alle estensioni**, ad esempio tramite l'aggiunta di librerie, ma **chiuso** (*Closed*) **alle modifiche**, ad esempio la sua modifica per adattarlo ad una applicazione.

Un modulo si dice **aperto alle estensioni** quando, ad esempio, viene usato all'interno di un altro modulo che aggiunge campi alle strutture dati del primo modulo o aggiunge ulteriori funzionalità, senza però modificare in alcun modo il modulo originale.

Un modulo si dice **chiuso alle modifiche** se è possibile usarlo in altri moduli senza che necessiti di essere modificato. Questo vuol dire che il modulo ha un'interfaccia stabile e ben definita che permette il suo riutilizzo "a scatola chiusa".

Encapsulation principle - Principio di incapsulamento

Read the following article in order to grasp the encapsulation principle in C language.

- [Encapsulation in C](#) by [Priya Pedamkar](#) hosted by [EDUCBA](#).
- [Answer](#) to [Data encapsulation in C](#) hosted by [stackoverflow](#).
- [Encapsulation \(computer programming\)](#) by [Wikipedia](#).

Esercizi

96. Progetta un'applicazione in grado di svolgere tutte le quattro operazioni fondamentali su due numeri **int** positivi **float**. Progetta una funzione distinta per ognuna delle quattro operazioni aritmetiche.

Indicazioni operative

- a. Prevedi l'input dei due valori nel `main.c` e, successivamente l'invocazione di ognuna delle quattro funzioni aritmetiche.
- b. Il risultato di ogni operazione deve essere visualizzato nel `main.c`.
- c. Per la moltiplicazione e la divisione effettua le seguenti prove:
 - i. crea una prima versione usando gli operatori aritmetici `*` e `/`
 - ii. crea una seconda versione creando un ciclo di somme per la moltiplicazione e un ciclo di sottrazioni per la divisione
 - iii. crea una terza versione invocando la funzione di somma per la moltiplicazione al posto di usare l'operatore `+` e, la funzione di sottrazione per la divisione al posto di usare l'operatore `-`
 - iv. crea una quarta versione usando la ricorsione → vedi il video [Recursion](#)
- d. Le quattro funzioni devono essere dichiarate nel file `operazioni.h` e dovrai definirle nel file `operazioni.c` → vedi il video [How To Create A Library And Split A Program Across Multiple Files](#)

97. Progettare un algoritmo in grado di svolgere le quattro operazioni fondamentali su due numeri, scegliendo tramite un menù l'operazione da svolgere.

Il programma presenta il seguente menù:

- 1) Somma
- 2) Sottrai
- 3) Moltiplica
- 4) Dividi
- 5) Esci

Scelta: _

e fatta la scelta chiede l'inserimento dei due operandi, visualizzando il risultato.

Indicazioni operative

- a. La visualizzazione del menù e l'inserimento della scelta devono essere gestite tramite una funzione da porre in un modulo separato, ad esempio `menu.c`, con le dichiarazioni in `menu.h`.
- b. Il menù deve essere visualizzato per scegliere l'operazione aritmetica fino a quando non si decide di terminare il programma.
- c. La scelta di una delle opzioni da 1 a 4 permette di eseguire l'operazione aritmetica richiesta visualizzando il risultato.
- e. Il risultato di ogni operazione aritmetica deve essere visualizzato nel `main.c`.
- d. Se viene scelta l'opzione 5 il programma deve terminare, salutando cortesemente l'utente.
- e. Se viene inserito un valore non valido per il menù si deve riproporre il menù stesso.
- f. Le quattro funzioni devono essere dichiarate nel file `operazioni.h` e dovrai definirle nel file `operazioni.c`.
- g. Scegli indipendentemente quando inserire i due valori su cui effettuare le operazioni aritmetiche, ma usa una funzione per il loro inserimento da definire in `operazioni.c` ed esporre in `operazioni.h`.

98. Progettare un algoritmo che determini in un gruppo di persone quante risultano essere sottopeso, quante normopeso, quante in sovrappeso e quante obese.

Si consideri che l'Indice di Massa Corporea (IMC) di una persona si determina con la seguente formula:

$$\text{IMC} = \text{peso (kg)} / \text{altezza}^2 (\text{m}^2)$$

e che l'Organizzazione Mondiale della Sanità individua le seguenti quattro categorie:

- a. sottopeso: IMC inferiore a 19;
- b. normopeso: IMC nell'intervallo [19, 24];
- c. sovrappeso: IMC nell'intervallo (24, 30];
- d. obeso: IMC superiore a 30.

Definisci delle macro per i limiti degli intervalli.

Usare le seguenti funzioni per gestire l'applicazione. Salvare le funzioni nel file `imc.c` e creare la propria libreria `imc.h` da importare nel file `main.c`.

- I. Inserimento e controllo dell'input, indipendentemente dal fatto che sia il numero di persone, il peso o l'altezza.
- II. Calcolo e restituzione dell'IMC.
- III. Conteggio nella giusta categoria del livello di IMC.
- IV. Visualizzazione dei conteggi sia come valori numerici (questa potrebbe non essere una funzione), sia come istogrammi orizzontali.

Domande

Rispondi alle seguenti domande per analizzare in modo [metacognitivo](#) la tua progettazione e implementazione dell'applicazione durante la sua creazione. Applica il metodo STAR (Stop, Think, Act, Review) 😊

1. Gli input dell'applicazione sono stati adeguatamente controllati? Individua correttamente il dominio di validità degli input e descrivi brevemente quali controlli hai previsto di fare. Se necessario aggiungi o modifica i controlli per rispettare i corretti domini di validità.
2. L'applicazione è stata separata in moduli diversi in modo che sia ben definita la competenza di ognuno di essi? Descrivi brevemente qual è la competenza generale di ogni modulo. Se il modulo ha troppe competenze sposta le funzioni nei moduli corrispondenti, eventualmente creandone dei nuovi.
3. Tutte le funzioni all'interno di un modulo sono fortemente coese e debolmente accoppiate? Descrivi queste caratteristiche relativamente alla tua applicazione. Se ci sono forti dipendenze tra funzioni riprogetta in modo che ci sia un basso accoppiamento. Se sono presenti delle funzioni che non rispondono alla competenza principale del modulo, spostale nel modulo corretto.
4. Sei sicuro che ogni funzione rispetti il principio di singola responsabilità? Se c'è più di una ragione per modificarla probabilmente ha troppe responsabilità e deve essere separata in più funzioni. Se necessario sistema la tua applicazione.

99. Realizzare un algoritmo che visualizzi il numero di giorni che intercorrono fra due date fornite in input, visualizzando un output analogo al seguente:

Data antecedente: 10/08/2012 Data successiva: 08/01/2016

Giorni che intercorrono fra le due date: 1246

Indicazioni operative

Prevedere le seguenti funzioni da gestire in file separati, raggruppandole in modo coeso in un modulo separato da quello della funzione `main()`.

- a. Funzione per l'inserimento e il controllo di una data.

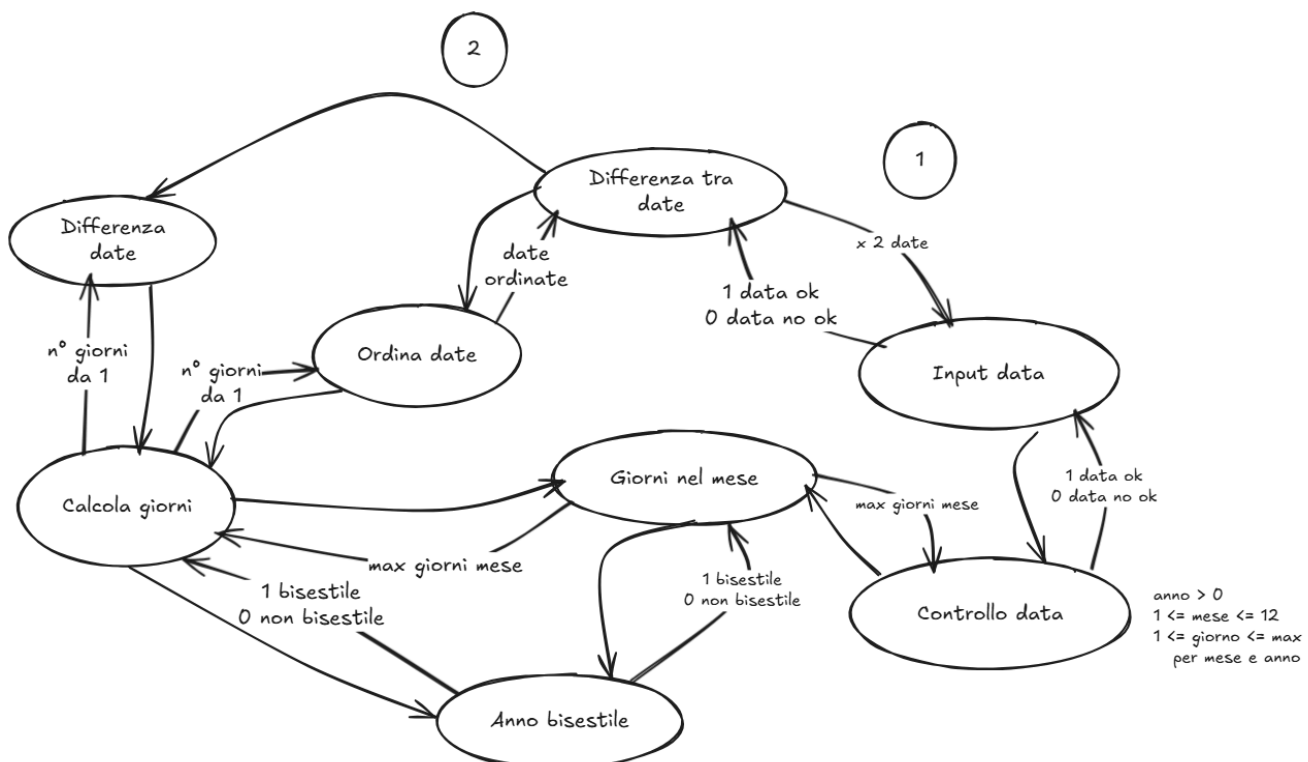
- b. Funzione che individua e fornisce in ordine la data antecedente e la data successiva.
- c. Funzione che determina se un anno è bisestile.
- d. Funzione che calcola il numero di giorni che intercorrono fra le due date e la relativa visualizzazione.
- e. Tutte le ulteriori funzioni di manipolazione delle date che ritieni siano utili. (← Pensa con cura come intendi effettuare la differenza tra date).

Domande

Rispondi alle seguenti domande per analizzare in modo [metacognitivo](#) la tua progettazione e implementazione dell'applicazione durante la sua creazione.

1. Gli input dell'applicazione sono stati adeguatamente controllati? Individua correttamente il dominio di validità degli input e descrivi brevemente quali controlli hai previsto di fare. Se necessario aggiungi o modifica i controlli per rispettare i corretti domini di validità.
2. L'applicazione è stata separata in moduli diversi in modo che siano ben definite le responsabilità di ognuno di essi? Descrivi brevemente qual è la responsabilità generale di ogni modulo. Se il modulo ha troppe responsabilità sposta le funzioni nei moduli corrispondenti, eventualmente creandone dei nuovi.
3. Tutte le funzioni all'interno di un modulo sono fortemente coese e debolmente accoppiate? Descrivi queste caratteristiche relativamente alla tua applicazione. Se ci sono forti dipendenze tra funzioni riprogetta in modo che ci sia un basso accoppiamento. Se sono presenti delle funzioni che non rispondono alla responsabilità principale del modulo, spostale nel modulo corretto.
4. Sei sicuro che ogni funzione rispetti il principio di singola responsabilità? Descrivi qual è la ragione per cui una funzione può essere modificata. Se c'è più di una ragione probabilmente deve essere separata in più funzioni. Se necessario sistema la tua applicazione.

Diagramma delle interazioni funzionali



100. Creare un'applicazione che realizzi il seguente gioco, il **Gioco dei tre numeri**, per tre giocatori.

Ogni giocatore deve estrarre casualmente tre numeri compresi tra 1 e 10 e il giocatore che ha la combinazione migliore aumenta il suo punteggio del valore della combinazione. I valori delle combinazioni sono così determinati:

- se c'è un tris (tre numeri uguali) la combinazione vale il numero che compare tre volte moltiplicato per 4;
- se c'è una coppia vale il numero che compare due volte moltiplicato per 2;
- se non ci sono tris o coppie il valore della combinazione è quello del numero più alto della combinazione.

Il gioco termina quando un giocatore raggiunge o supera i 100 punti e, in ogni caso termina dopo 5 round. In quest'ultimo caso, se nessuno ha raggiunto i 100 punti, non esiste un vincitore.

Indicazioni operative

Consegnare il diagramma delle interazioni funzionali dell'applicazione. Per redigerlo puoi usare lo strumento online [excalidraw](#), oppure un qualsiasi altro strumento che ritieni funzionale. Carta e penna vanno benissimo, fai una foto e la carichi nel documento di consegna.

Prevedere le seguenti funzioni da gestire in file separati, raggruppandole in modo coeso in moduli separati da quello della funzione main().

- a. Funzione per l'estrazione di un numero casuale in un qualsiasi intervallo.
- b. Funzione per il calcolo del valore della combinazione secondo le regole fornite.
- c. Funzione (o funzioni) per giocare un turno per un dato giocatore. La funzione deve effettuare le tre estrazioni, visualizzarle per ogni giocatore (si distinguano i giocatori con una stringa passata alla funzione) e visualizzare il valore finale della combinazione corrente. Si usino le funzioni dei punti (a) e (b).
- d. Funzione per la visualizzazione dei punteggi per ogni turno o del punteggio finale. Si distinguano le visualizzazioni con una stringa passata alla funzione.
- e. La funzione che visualizza il vincitore o la segnalazione che nessuno ha vinto dopo 5 round perché nessun giocatore ha raggiunto i 100 punti.
- f. Usa gli [header guard](#) nel file .h.

Per la generazione del numero pseudo casuale si utilizzino le funzioni `time()` per recuperare il tempo corrente del sistema, `srand()` per impostare il "seme", cioè il punto di inizio per la generazione del numero pseudo casuale, e `rand()` per la generazione effettiva del numero pseudo casuale in un dato intervallo. Di seguito viene fornito un esempio per la generazione di un numero pseudo casuale:

```
srand(time(0)); // Imposta il tempo corrente del sistema come seme per la
                // generazione del numero pseudo casuale
int num = (rand() % (UPPER - LOWER + 1)) + LOWER;
// Genera un numero pseudo casuale nell'intervallo [LOWER - UPPER). La
funzione rand() genera un numero pseudo casuale nell'intervallo [0 - 1),
```

l'uso dei valori LOWER e UPPER permettono di variare l'intervallo di generazione

101. Un panificio ha tre fornai strepitosi, ognuno specializzato nella preparazione di un particolare prodotto: il primo prepara il pane, il secondo la pasta e il terzo la pizza.

I tre fornai, quando iniziano a preparare aprono un sacco di farina che usano tutti insieme. Un sacco contiene 5 kg di farina, mentre una forma di pane necessita di 100 gr di farina, per una porzione di pasta servono 75 gr di farina e per una teglia di pizza servono 80 gr di farina.

Ad ogni fornaio vengono richiesti un numero variabile di pezzi della propria specialità e, se nel sacco c'è farina sufficiente, prepara i pezzi richiesti, altrimenti prepara ciò che può con la farina disponibile.

Un cliente che arriva al forno ordina il prodotto desiderato usando un menù e inserisce per ogni prodotto anche il numero di pezzi richiesti. Il menù permette anche di terminare le ordinazioni.

Progetta un'applicazione che permetta di fare le ordinazioni desiderate fino a quando non si decide di chiudere il forno, terminando l'applicazione. Per ogni ordine deve essere visualizzato cosa effettivamente il fornaio è riuscito a produrre.

Indicazioni operative

Creando l'applicazione seguire quanto indicato di seguito. Le funzioni devono essere gestite in file separati, raggruppandole in modo coeso in un modulo separato da quello della funzione main().

- a. Gestisci il sacco di farina come una [variabile statica](#).
- b. Gestisci i grammi di farina per ogni prodotto come [macro](#).
- c. Prevedi una sola funzione per i tre fornai che si deve distinguere tramite una stringa costante passata come parametro e i grammi necessari per il proprio prodotto, oltre al numero di prodotti.
- d. Gestisci il menù tramite una funzione.
- e. Gestisci con una funzione la visualizzazione di cosa è stato prodotto.
- f. Usa gli [header guard](#) nel file .h.

102. Il tachimetro per biciclette determina la velocità espressa in metri/minuto a partire dal numero g di giri della ruota conteggiati in un periodo di 10 secondi in base alla seguente formula:

$$v = 3,1416 \cdot d \cdot g \cdot 6$$

dove d è il diametro della ruota espresso in metri.

Progettare un algoritmo che, a partire dal diametro della ruota e dal numero di giri conteggiati in 10 secondi, determini la velocità in km/h della bicicletta.

103. Progettare una funzione che determini il MCD (massimo comune divisore) tra due numeri interi m ed n (con $m > n$) utilizzando il seguente algoritmo attribuito ad Euclide:

- a. determinare il resto r della divisione tra m ed n ;
- b. se r è uguale a 0 terminare: il massimo comune divisore tra m ed n è n ;
- c. altrimenti porre m uguale al valore di n , porre n uguale al valore di r e riprendere dal passo a.

Utilizzare la funzione descritta per determinare il MCD di due numeri forniti in input.

104. Progettare un algoritmo che, a partire dal valore di due parametri interi che rappresentano rispettivamente il numeratore e il denominatore di una frazione, li modifichi in modo da rendere la frazione risultante ridotta ai minimi termini. Si ricordi che la riduzione di una frazione ai minimi termini si ottiene dividendo sia il numeratore, sia il denominatore per il loro MCD.
105. Cosa fa il seguente programma? E come fa a farlo?

Prima versione

```
#include <stdio.h>

#define ROWS 5

void carica(int*);
void visualizza(int*);

int main(void) {
    int numeri[ROWS];

    carica(numeri);
    visualizza(numeri);
    return 0;
}

void carica(int *valori) {
    for (int i = 0; i < ROWS; i++) {
        printf("Inserisci il valore %d all'indirizzo %p: ", i+1, valori);
        scanf("%d", valori++);
    }
}

void visualizza(int* valori) {
    printf("Valori inseriti:\n");
    for (int i = 0; i < ROWS; i++) {
        printf("Valore %d all'indirizzo %p: %d\n", i+1, valori, *valori);
        valori++;
    }
}
```

Seconda versione

```
#include <stdio.h>

#define ROWS 5

void carica(int valori[]);
void visualizza(int valori[]);

int main(void) {
```

```
int numeri[ROWS];

carica(numeri);
visualizza(numeri);
return 0;
}

void carica(int valori[]) {
    for (int i = 0; i < ROWS; i++) {
        printf("Inserisci il valore %d all'indirizzo %p: ", i+1,
&valori[i]);
        scanf("%d", &valori[i]);
    }
}

void visualizza(int valori[]) {
    printf("Valori inseriti:\n");
    for (int i = 0; i < ROWS; i++) {
        printf("Valore %d all'indirizzo %p: %d\n", i+1, &valori[i],
valori[i]);
    }
}
```

106. bla bla

Compilazione e interpretazione dei programmi

C Compiler⁶

La compilazione è il processo di conversione in codice macchina del codice sorgente scritto in linguaggio C. Poiché il C è un linguaggio compilato, ha bisogno di un compilatore per trasformarlo in codice eseguibile, in modo che il programma possa essere eseguito sulla nostra macchina.

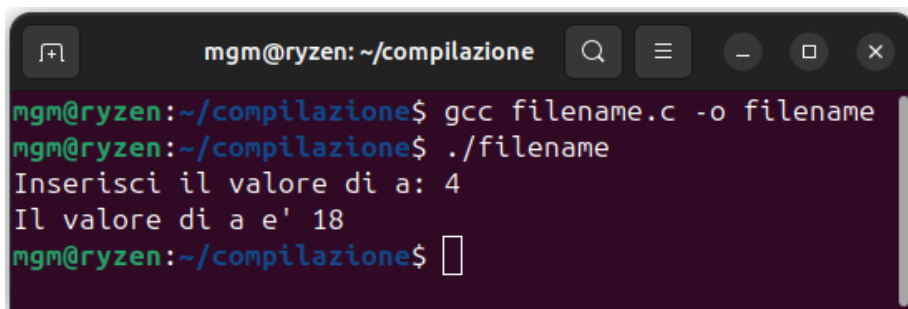
Un file sorgente scritto in linguaggio C per essere compilato e creare il corrispondente file eseguibile utilizza la seguente istruzione direttamente scritta sul terminale o eseguita dal proprio IDE.

```
$ gcc filename.c -o filename
```

Se si lavora in ambiente Linux esecuzione del file eseguibile può avvenire attraverso la seguente istruzione:

```
$ ./filename
```

Il risultato che si ottiene è praticamente il seguente supponendo che il programma **filename.c** debba svolgere un semplice calcolo.

A screenshot of a terminal window with a dark background. The title bar shows 'mgm@ryzen: ~/compilazione'. The terminal content shows the following commands and output:

```
mgm@ryzen:~/compilazione$ gcc filename.c -o filename
mgm@ryzen:~/compilazione$ ./filename
Inserisci il valore di a: 4
Il valore di a e' 18
mgm@ryzen:~/compilazione$
```

filename.c

```
#include <stdio.h>
#define VALORE 6

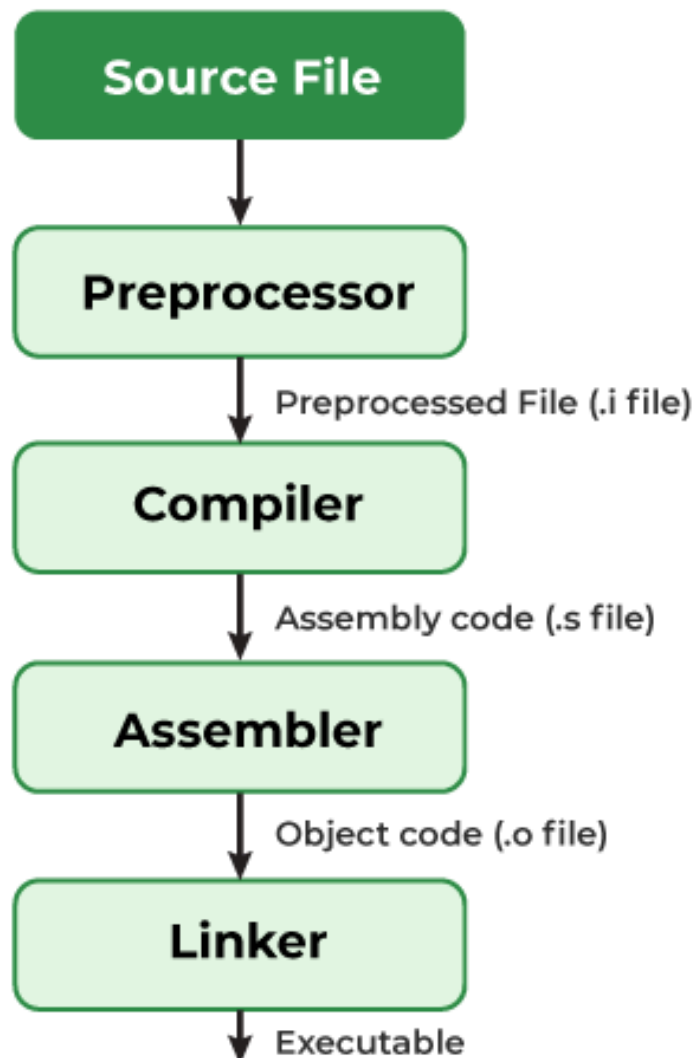
int main() {
    int a;
    const int b = 3;

    printf("Inserisci il valore di a: ");
    scanf("%d", &a);
    a = a * b + VALORE;
    printf("Il valore di a e' %d\n", a);
    return 0;
}
```

⁶ [Compiling a C Program: Behind the Scenes](#) by [GeeksforGeeks](#)
[Understanding the C Compilation Process | Step-by-Step Tutorial](#) by [Micro&Coding](#)

Per creare il file eseguibile il compilatore deve convertire il file sorgente in linguaggio macchina, svolgendo le operazioni indicate nell'immagine seguente.

La prima fase viene svolta dal precompilatore o pre-processor, successivamente interviene il compilatore, poi l'assemblatore e infine il linker. Tutti questi elementi contribuiscono alla creazione del file eseguibile.



Preprocessing

Le operazioni svolte dal precompilatore (pre-processor) sono la prima fase della compilazione. Il precompilatore svolge le seguenti operazioni:

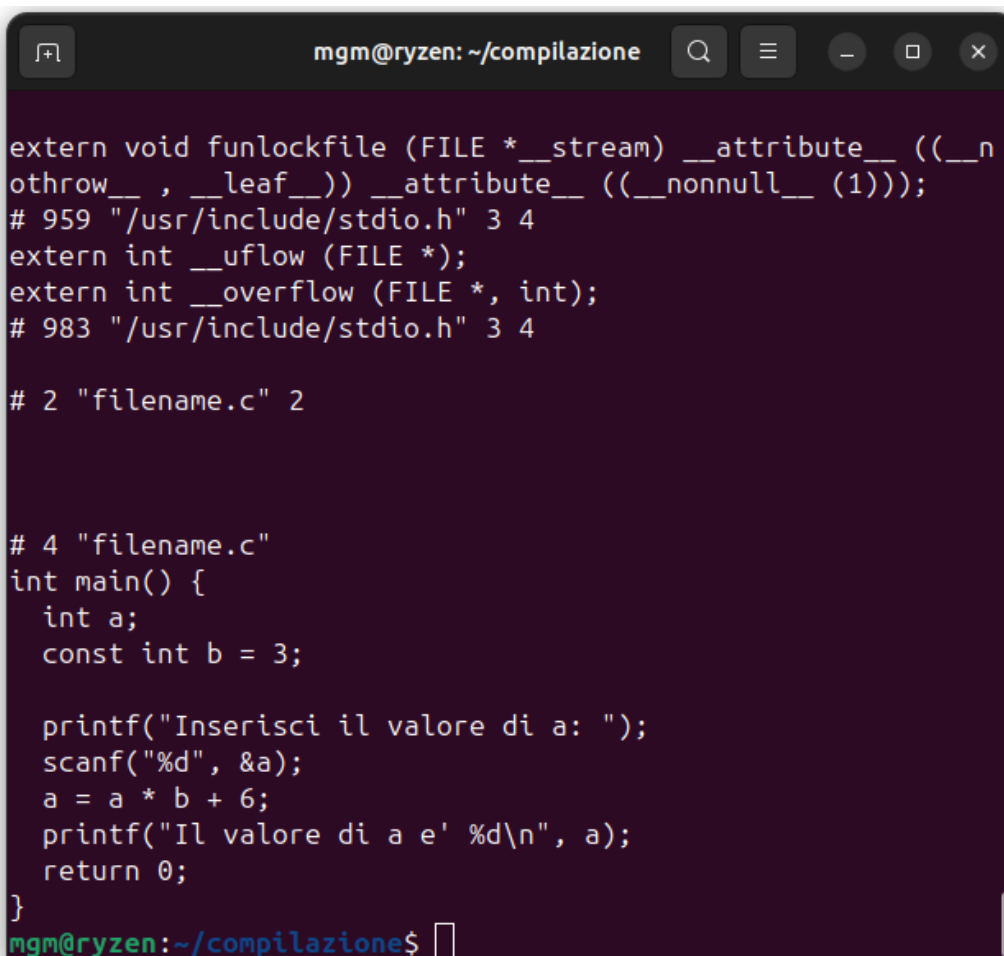
- rimuove tutti i commenti;
- espande le macro definite con la direttiva `#define`;
- espande tutti i file inclusi con la direttiva `#include`;
- gestisce e risolve tutte le compilazioni condizionali definite attraverso la [direttiva `#ifdef`](#).

L'output della precompilatore viene visualizzato immediatamente o memorizzato in un file con estensione `.i` se esplicitamente richiesto come mostrato nel paragrafo [Riepilogo dei comandi per la compilazione a linea di comando](#). Verificando il contenuto di questo file, vengono trovate tutte le dichiarazioni delle funzioni previste dalle direttive `#include` dell'applicazione.

È possibile fermare la traduzione di un file sorgente al livello del precompilatore usando il seguente comando:

```
$ gcc -E filename.c
```

L'ultimo frammento dell'azione del precompilatore sul file `filename.c` è il seguente.



```
mgm@ryzen: ~/compilazione
extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
# 959 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 983 "/usr/include/stdio.h" 3 4

# 2 "filename.c" 2


# 4 "filename.c"
int main() {
    int a;
    const int b = 3;

    printf("Inserisci il valore di a: ");
    scanf("%d", &a);
    a = a * b + 6;
    printf("Il valore di a e' %d\n", a);
    return 0;
}
mgm@ryzen:~/compilazione$
```

Compiling

Il passo successivo dell'intero processo di traduzione del file sorgente è la compilazione del file generato dal precompilatore (se è stato creato è il file con estensione **.i**). La compilazione produce un file con estensione **.s** composto dalla traduzione del file sorgente in linguaggio assembly.

È possibile fermare la traduzione di un file sorgente al livello del compilatore usando il seguente comando:

```
$ gcc -S filename.c -o filename.s
```

Ciò che si ottiene è un file analogo all'immagine di lato.

Assembling

In questa fase il file **.s** viene tradotto in un file **.o** dall'assembler, detto **file oggetto**. Questo file contiene istruzioni in linguaggio macchina.

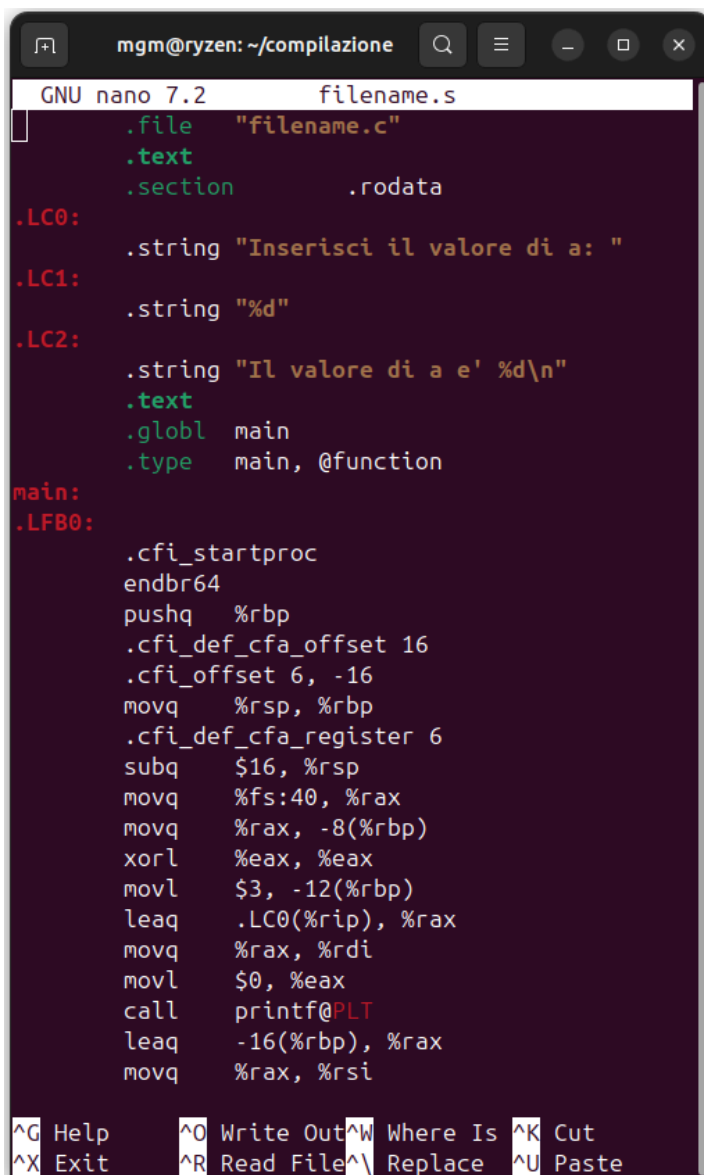
In questa fase viene tradotto in linguaggio macchina solo il codice che riguarda il nostro programma, senza includere tutte le funzioni esterne, come la `printf()` o la `scanf()`.

È possibile fermare la traduzione di un file sorgente al livello di assembling usando il seguente comando:

```
$ gcc -c filename.c
```

Il file **filename.o** viene generato automaticamente.

Il file **.o** non è visualizzabile in quanto è scritto in codice macchina. L'apertura del file **.o** con un editor di testi tenta, senza riuscirci, di visualizzare il codice binario.



```
mgm@ryzen: ~/compilazione
GNU nano 7.2 filename.s
.file "filename.c"
.text
.section .rodata
.LC0:
.string "Inserisci il valore di a: "
.LC1:
.string "%d"
.LC2:
.string "Il valore di a e' %d\n"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movq %fs:40, %rax
movq %rax, -8(%rbp)
xorl %eax, %eax
movl $3, -12(%rbp)
leaq .LC0(%rip), %rax
movq %rax, %rdi
movl $0, %eax
call printf@PLT
leaq -16(%rbp), %rax
movq %rax, %rsi
```

[illegible]

Linking

Questa è la fase finale in cui viene effettuato il collegamento di tutte le chiamate di funzione con le loro definizioni. Il linker sa dove sono implementate tutte le funzioni come la `printf()` o la `scanf()`, o le funzioni personali e, unisce tutto per creare l'eseguibile finale. Anche in questo caso il contenuto del file generato non è visualizzabile.

Il comando per terminare la compilazione fino al linker genera il file eseguibile nel seguente modo:

```
$ gcc filename.c -o filename
```

Il linker aggiunge anche del codice ulteriore al programma necessario per l'avvio e la chiusura dell'applicazione. Questo codice aggiuntivo può riguardare la configurazione dell'ambiente dell'applicazione tramite dei parametri iniziali, o altro codice necessario per il suo corretto funzionamento.

L'aggiunta delle funzioni usate dall'applicazione e del codice aggiuntivo di configurazione è verificabile usando i comandi `$size filename.o` e `$size filename`. Attraverso questi comandi è possibile osservare la diversa dimensione dei due file: il file finale **filename** è decisamente più grande del file oggetto **filename.o**. Questo aumento è dovuto al codice aggiuntivo che il linker inserisce nel programma.

```

mgm@ryzen: ~/compilazione
mgm@ryzen:~/compilazione$ size filename.o
  text    data    bss     dec     hex filename
   286      0      0     286    11e filename.o
mgm@ryzen:~/compilazione$ size filename
  text    data    bss     dec     hex filename
 1780    616      8   2404    964 filename
mgm@ryzen:~/compilazione$

```

Riepilogo dei comandi per la compilazione a linea di comando

Creazione di un file per **ogni fase della compilazione** (filename.i, filename.s, filename.o, filename):

```
$ gcc -Wall -save-temps filename.c -o filename
```

Creazione del solo file fino alla fase svolta dal **Precompilatore** (filename.i):

```
$ gcc -E filename.c
```

Creazione del solo file fino alla fase svolta dal **Compiler** (filename.s):

```
$ gcc -S filename.c -o filename.s
```

Creazione del solo file oggetto fino alla fase svolta dall'**Assembler** (filename.o):

```
$ gcc -c filename.c
```

Creazione del solo file eseguibile fino alla fase svolta dal **Linker** (filename):

```
$ gcc filename.c -o filename
```

Compiler vs Interpreter vs Hybrid compiling

Read the following article and watch the videos to grasp the difference between compiler, interpreter, and hybrid compiling.

- [Difference between Compiler and Interpreter](#) by [GeeksforGeeks](#)
- [Compiler and Interpreter: Compiled Language vs Interpreted Programming Languages](#) by [Coding Mentors](#)

La struttura della memoria di un processo⁷

In questo paragrafo viene analizzata la **memoria virtuale di un processo**, supponendo che i programmi vengano mandati in esecuzione su un sistema operativo Linux multitasking installato su una architettura x86 a 32-bit.

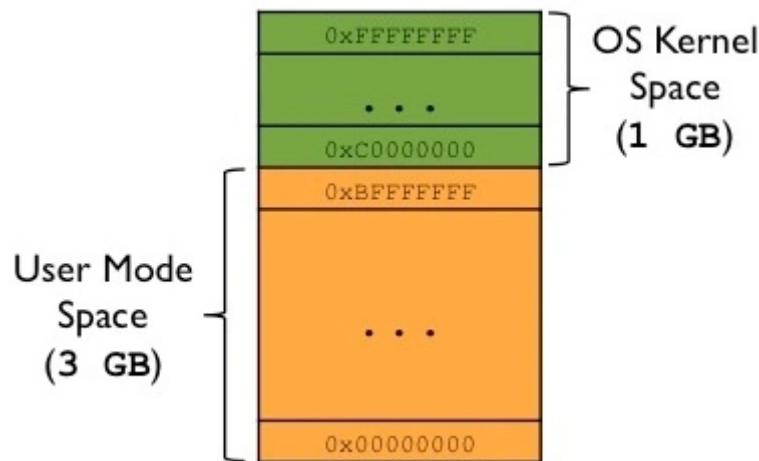
In un sistema operativo multitasking ogni processo viene eseguito nel proprio spazio di indirizzamento (*sandbox*).

Dato che la minima unità indirizzabile da una CPU è 1 byte (8 bit), una CPU a 32-bit è in grado di generare 2^{32} numeri diversi che corrispondono agli indirizzi di ogni singolo byte presente in memoria. Quindi in un'architettura a 32-bit lo spazio virtuale indirizzabile totale della memoria è di 4GB (2^{32}) blocchi di memoria.

Questi indirizzi virtuali generati dalla CPU devono poi essere mappati sugli indirizzi fisici reali tramite l'uso della tabella delle pagine mantenuta dal kernel del sistema operativo.

Dato che il kernel è anch'esso un processo, gli sarà dedicata una porzione dello spazio di indirizzamento virtuale, separata dalla porzione dedicata per i processi utente. Ad esempio, i primi 3 GB della memoria virtuale possono essere dedicati ai processi utente (dall'indirizzo 0x00000000 all'indirizzo 0xBFFFFFFF), mentre l'ultimo GB può essere dedicato al kernel del sistema operativo (dall'indirizzo 0xC0000000 all'indirizzo 0xFFFFFFFF).

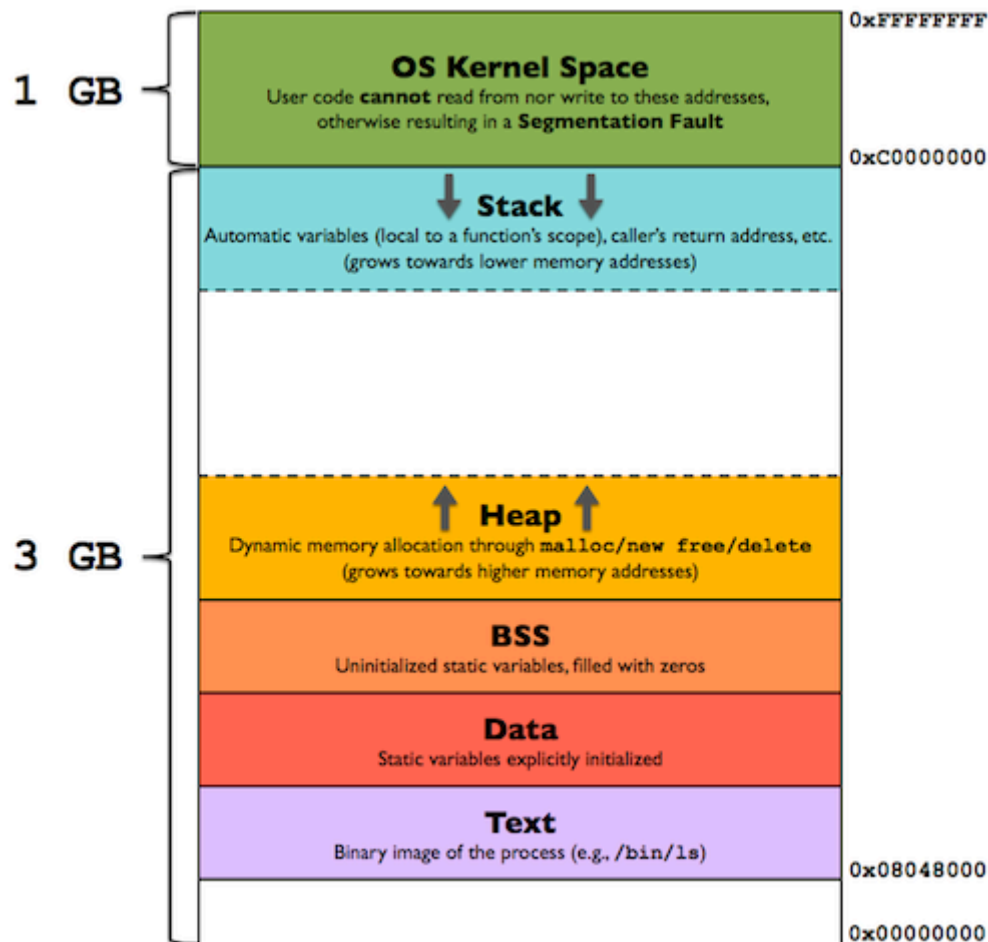
La seguente immagine mostra una possibile suddivisione della memoria virtuale tra la modalità user e la modalità kernel in un tipico sistema Linux a 32-bit.



⁷ [Gabriele Tolomei, In-Memory Layout of a Program \(Process\)](#), ultimo accesso febbraio 2022

La memoria virtuale di un processo

La **memoria virtuale di un processo**⁸ viene divisa in segmenti, cioè un insieme contiguo di indirizzi virtuali ai quali il processo può accedere. Solitamente un programma C viene suddiviso nei seguenti segmenti, cioè ha un proprio spazio di indirizzamento privato e non condiviso con altri processi e, tutti i processi in esecuzione gestiscono il proprio spazio di memoria allo stesso modo.



Text - Il segmento *Text*, conosciuto anche come segmento *Code*, è la sezione di memoria che contiene le **istruzioni eseguibili** del programma, **in linguaggio macchina**.

Le istruzioni corrispondono al codice del programma, al codice delle funzioni di librerie usate dal programma e alle costanti.

Questa sezione viene marcata in sola lettura per evitare sovrascritture accidentali o maliziose che ne modifichino le istruzioni e, viene allocata nella memoria all'avvio del programma da una funzione particolare del kernel chiamata `exec`.

Questa sezione rimane invariata per tutto il tempo dell'esecuzione del programma e, per questa ragione il segmento *Text* è condivisibile tra più programmi in modo tale che ne venga mantenuta una sola copia in memoria anche se usata da più processi dello stesso programma.

⁸ Simone Piccardi (<http://piccardi.gnulinix.it/>), *GaPiL - Guida alla Programmazione in Linux* (<http://gatil.gnulinix.it/files/share/gatil.pdf>): 2.2.2 La struttura della memoria di un processo (<http://gatil.gnulinix.it/files/2011/12/gatil.pdf#subsection.2.2.2>)

Per capire l'utilità di usare più volte la stessa porzione di codice, si pensi ad un text editor in cui si possono aprire più documenti contemporaneamente, oppure ad un compilatore che può compilare più programmi, o ancora alla shell che può essere aperta su più finestre e così via. Il codice eseguibile viene caricato in memoria una sola volta, ma più esecuzioni parallele o concorrenti ([thread](#)) dello stesso codice possono avvenire contemporaneamente. Questa è la ragione per cui è possibile, ad esempio, ascoltare la musica dal proprio dispositivo e contemporaneamente modificare il volume. Se non ci fossero esecuzioni contemporanee di porzioni dello stesso codice, questa operazione non sarebbe possibile. Il codice viene caricato in memoria una sola volta e, thread diversi vengono mandati in esecuzione.

Data - Nel segmento *Data* sono presenti i *dati statici o globali espressamente inizializzati* dal programmatore, cioè le variabili globali (quelle definite al di fuori di tutte le funzioni che compongono il programma) e le [variabili statiche](#) (quelle dichiarate con l'attributo `static`).

Questo segmento è preallocato all'avvio del programma e inizializzato ai valori specificati nelle dichiarazioni delle variabili globali e statiche.

Il segmento *Data* può essere ulteriormente suddiviso nell'area dei dati globali inizializzati e in sola lettura (*read-only*) e, nell'area dei dati globali inizializzati modificabili (*read-write*). Ad esempio, supponiamo di avere le seguenti dichiarazioni globali in un programma C.

```
char *hello = "hello world";
int debug = 1;
const char* str = "hello world again";

void fun() {
    static int i = 10;
    ...
}

int main() {
    ...
    fun();
    ...
    return 0;
}
```

Le due variabili `char *hello` e `int debug` sono memorizzate nell'area dei dati globali inizializzati modificabili (*read-write*).

La variabile costante `const char* str` definisce la variabile `str` puntatore (*) a `char` che punta al primo carattere della stringa letterale **costante "hello world again"**. In questo caso **la stringa letterale costante "hello world again" è memorizzata nell'area dei dati globali inizializzati in sola lettura** (*read-only*) vista la presenza del modificatore `const`, mentre **la variabile `str` è memorizzata nell'area dei dati globali inizializzati modificabili** (*read-write*), in quanto il suo contenuto può essere modificato a run-time, modificando l'indirizzo di memoria memorizzato nella variabile puntatore `str`. Il modificatore `const` posto prima del tipo rende costante solo l'area di memoria puntata da `str`.

La variabile `static int i` prevede anch'essa la memorizzazione della variabile `i` nell'area dei dati globali inizializzati modificabili (*read-write*). La presenza del modificatore `static` rende permanenti le modifiche apportate alla variabile `i` in successive invocazioni della funzione `fun()` e, per poterlo fare, è necessario che la variabile `i` non sia memorizzata nello stack frame della funzione `fun()`, deve essere salvata in un'area di memoria che non venga eliminata quando lo stack frame della funzione `fun()` viene eliminato. Il segmento *Data* funge allo scopo.

BSS - Il segmento *BSS* contiene le *variabili globali o statiche non esplicitamente inizializzate* dal programmatore, e il suo nome deriva da un vecchio operatore del linguaggio assembly nominato *Block Started by Symbol*.

Di solito questo segmento inizia alla fine del segmento *Data*, i dati del segmento *BSS* sono sempre modificabili (*read-write*) e vengono inizializzati dal kernel al valore 0 prima che il programma venga eseguito.

Il seguente esempio prevede un array globale immagazzinato nel segmento *BSS*:

```
int vect[100];

int main() {
    ...
    return 0;
}
```

I puntatori salvati nel segmento *BSS* sono inizializzati a `NULL`, ma questo comportamento viene tenuto solo all'interno di questo segmento, mentre in generale un puntatore non è inizializzato preventivamente dal kernel.

Stack - Nel segmento *Stack* vengono memorizzati i valori delle variabili locali e dei parametri formali delle funzioni del programma che sono state mandate in esecuzione.

In una architettura x86 il segmento *Stack* è tipicamente collocato nella parte degli indirizzi più grandi dello spazio di indirizzamento virtuale del processo e cresce verso gli indirizzi più piccoli, in modo opposto al verso di crescita del segmento *Heap*. In altre architetture può crescere nella direzione opposta.

Il segmento *Stack* è organizzato come una struttura LIFO (*Last In First Out*) in modo tale che in cima alla pila siano sempre presenti i dati (*stack frame*) della funzione attualmente in esecuzione (variabili locali, parametri formali, indirizzo di ritorno alla funzione chiamante). Successivamente sono presenti i dati (*stack frame*) delle funzioni che sono state invocate precedentemente, ma che hanno interrotto la loro esecuzione per l'invocazione di un'altra funzione.

L'ordine degli *stack frame* delle funzioni all'interno dello *Stack* è esattamente quello di invocazione delle funzioni. In cima allo *Stack* è sempre presente lo *stack frame* della funzione attualmente in esecuzione. Tutti i dati di una funzione vengono impilati in sequenza uno sull'altro in modo tale che le funzioni possano essere chiamate ricorsivamente "impilando" un nuovo *stack frame* sulla cima dello *Stack* e, rendendo l'insieme delle variabili di una chiamata di funzione completamente indipendente da quello di un'altra chiamata di funzione. Quando una funzione termina, il suo *stack frame* viene automaticamente rilasciato (*deallocated*).

Lo **Stack Pointer Register** della CPU contiene l'indirizzo di memoria della cima dello Stack e viene aggiornato ogni volta che viene inserito un nuovo valore nello *Stack*. Se il valore contenuto nello *Stack Pointer Register* dovesse raggiungere la cima dello *Heap*, o dovesse raggiungere il limite imposto dal sistema operativo memorizzato nella variabile di sistema `RLIMIT_STACK`, verrebbe segnalato un errore di *Stack Overflow*.

Heap - Il segmento *Heap* può essere considerato come un'estensione dei segmenti contenenti i dati. In questo segmento avviene l'allocazione dinamica della memoria, cioè *l'allocazione delle variabili la cui dimensione può essere conosciuta solo a run-time e non può essere determinata statisticamente dal compilatore prima dell'esecuzione del programma*.

Il segmento *Heap* inizia subito dopo il segmento *BSS* e cresce verso gli indirizzi alti di memoria. Viene gestita tramite le istruzioni `malloc/new` e `free/delete` che usano le system call `brk` e `sbrk`⁹ per regolare la sua dimensione.

⁹ Le system call `brk` e `sbrk` sono delle system call di base per la gestione della memoria, usate nei sistemi operativi Unix e Unix-like per controllare la quantità di memoria allocata nel segmento *Data* da un processo. Queste funzioni sono di solito invocate da una funzione di libreria di gestione della memoria di più alto livello, come la `malloc`. [Wikipedia](#) - [sbrk](#).

The Call Stack

Watch the following video in order to understand properly how stack works.

- [The Call Stack](#) by [Computer Science](#)
- [The Call Stack](#) by [MrBrownCS](#)
- [Stack vs Heap Memory - Simple Explanation](#) by [Alex Hyett](#)

Learn more about Stack and Heap

Read the following article for an in-depth analysis about Stack and Heap.

- [x86/x64 CPU architecture: the stack & stack frames](#) by [Yuriy Georgiev](#)

Strutture dati

Array e stringhe

C Tutorial - Array

10.C Programming **Arrays** by [Programiz](#)

- a. [C Arrays](#)
- b. [C Multidimensional Arrays](#)
- c. [Pass arrays to a function in C](#)
- d. [Relationship Between Arrays and Pointers](#)

11.VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)

- a. [Array Basics](#)
- b. [Passing an Array to a Function](#)
- c. [Constants](#)
- d. [Find The Length Of An Array Using sizeof\(\) | C Programming Tutorial](#) ←
IMPORTANTE!!! Per ora puoi fermarti al minuto 13:54, la funzione malloc() la vedremo più avanti
- e. [2D Array Basics](#)
- f. [Pointer Notation](#)
- g. [How To Return An Array From A Function | C Programming Tutorial](#) ←
IMPORTANTE!!! Per ora puoi fermarti al minuto 8:03, la funzione malloc() la vedremo più avanti
- h. [array vs &array Pointers Difference Explained](#) come approfondimento
- i. [Function-like Macros](#) come approfondimento

C Tutorial - String

12.C Programming **Strings** by [Programiz](#)

- a. [C Programming Strings](#)
- b. [String Manipulations In C Programming Using Library Functions](#)
- c. [C strlen\(\)](#)
- d. [C strcpy\(\)](#)
- e. [C++ strncpy\(\)](#)
- f. [C++ sprintf\(\)](#)
- g. [C strcmp\(\)](#)
- h. [C strcat\(\)](#)

13.VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)

- a. [String Basics](#)
- b. [Handling String Input With Spaces](#)
- c. [strcpy\(\) and strncpy\(\) functions](#)
- d. [sprintf\(\) Function | C Programming Tutorial](#)
- e. [How To Return A String From A Function | C Programming Tutorial](#) ←
IMPORTANTE!!! Per ora puoi fermarti al minuto 9:38, la funzione malloc() la vedremo più avanti
- f. [strcmp\(\) function](#)

- g. [Convert A String To A Double, Int Or Long Int With atof\(\), atoi\(\) or atol\(\)](#)
- h. [String In Char Array VS. Pointer To String Literal](#)

14. Dimensionamento dello spazio di visualizzazione di una stringa:

<https://stackoverflow.com/questions/1809399/how-to-format-strings-using-printf-to-get-equal-length-in-the-output>

Esempio sull'uso di array e stringhe

Array numerici con singola stringa (es.122 e es.124)

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib/studente.h"

#define LEN 10
#define DIM 3
#define RIS 14

int main() {
    char nome[LEN];
    float voti[DIM];
    float media = 0.;
    char risultato[RIS];

    /*
    1) Inserisci studente
    2) Carica voti
    3) Visualizza voti
    4) Visualizza stato studente
    5) Esci
    Scelta: _

    VINCOLI
    Se manca 1 gli altri punti non possono
        essere eseguiti
    Se manca 2 non potete eseguire 3 e 4
    */

    // Input nome
    datiStudente("Nome studente: ", nome, LEN);
    // Input voti
    caricaVoti("Inserire il %d voto: ", voti, DIM);
    // Calcolo media
    media = mediaVoti(voti, DIM);
    // Visualizza voti
    visualizzaVoti(nome, voti, DIM);
    printf("La media dei voti di %s e': %.2f\n", nome, media);
    determinaStato(risultato, media);
    printf("%s e' %s\n", nome, risultato);
    return 0;
}
```

studente.h

```
#pragma once
void datiStudiante(const char *messaggio, char *nome, int len);
void caricaVoti(const char *messaggio, float *voti, int dim);
float mediaVoti(float *voti, int dim);
void visualizzaVoti(char *nome, float *voti, int dim);
void determinaStato(char *risultato, float media);
```

studente.c

```
#define DIM_BUF 100

/**
 * Carica il nome dello studente
 */
void datiStudiante(const char *messaggio, char *nome, int len) {
    printf("%s", messaggio);
    scanf("%s", nome);
    nome[len-1] = '\0';
    while(getchar() != '\n');
}

/**
 * Carica i voti dello studente
 */
void caricaVoti(const char *messaggio, float *voti, int dim) {
    int votoOk ;
    for(int i = 0; i < dim; ++i) {
        char buffer[DIM_BUF];
        sprintf(buffer, messaggio, i+1);
        printf("%s", buffer);
        votoOk = scanf("%f", &voti[i]);
        while(votoOk != 1 || (voti[i] < 1 || voti[i] > 10)) {
            printf("Errore\n");
            printf("%s", buffer);
            votoOk = scanf("%f", &voti[i]);
        }
    }
}

/**
 * Calcola la media dei voti dello studente
 */
float mediaVoti(float *voti, int dim) {
    float media = 0.;
    for(int i = 0; i < dim; ++i) {
        media += voti[i];
    }
    media = media / dim;
    return media;
}

/**
 * Visualizza i voti dello studente
 */
void visualizzaVoti(char *nome, float *voti, int dim) {
```

```
printf("I voti di %s sono i seguenti:\n", nome);
for(int i = 0; i < dim; ++i) {
    printf("Voto %d: %.2f\n", i+1, voti[i]);
}

}

/**
 * Determina lo stato sufficiente o insufficiente dello studente
 */
void determinaStato(char *risultato, float media) {
    if (media < 6) {
        strcpy(risultato, "insufficiente");
    } else {
        strcpy(risultato, "sufficiente");
    }
}
```

Array paralleli stringhe con elenco di singoli numeri

Obiettivo - Il seguente esempio permette il caricamento dei nomi delle città e dei valori di PM10 per ognuna di esse. L'applicazione, oltre al caricamento dei dati, permette di visualizzare i nomi delle città che hanno un valore di PM10 pari al massimo e di visualizzare genericamente lo stato di tutte le città.

main.c

```
/**
 * Città e PM10
 */
#include <stdio.h>
#include <stdlib.h>
#include "lib/menu.h"
#include "lib/citta.h"

#define NUM_CITTA 5
#define LUNGHEZZA 20
#define NUM_OPZIONI 4

int main() {
    char citta[NUM_CITTA][LUNGHEZZA];
    float particolato[NUM_CITTA];
    int scelta;
    char *opzioni = "1) Carica citta' con particolato\n"
                   "2) Citta' con PM10 massimo\n"
                   "3) Visualizza citta' e particolato\n"
                   "4) Esci\n"
                   "Scelta: ";

    scelta = menu(opzioni, NUM_OPZIONI);
    while (scelta != NUM_OPZIONI) {
        switch(scelta) {
            case 1:
                /*Caricamento delle singole città e del corrispondente
                 valore di PM10*/
                caricaDati(NUM_CITTA, LUNGHEZZA, citta, particolato);
                break;
            case 2: ;
                /*Le due istruzioni commentate permettono di detrminare
                 la posizione del singolo valore massimo, il primo che
                 viene trovato, e di visualizzare il nome della città
                 corrispondente.*/
                //posizione = posizioneMassimo(NUM_CITTA, particolato);
                //printf("La citta' con il PM10 massimo a' %s\n", citta[posizione]);

                /*Le successive istruzioni permettono di determinare le
                 posizioni di tutte le città che hanno un valore pari al
                 massimo e di visualizzare tutti i loro nomi.*/
                int posizioniMax[NUM_CITTA]; //Array che contiene le posizioni
                                           //dei massimi
                //Funzione che salva in posizioniMax le posizioni dei massimi
                posizioniMassime(NUM_CITTA, particolato, posizioniMax);
                //Funzione che visualizza tutte e sole le città con i valori
                //massimi di PM10
```

```

        visualizzaMassimi(NUM_CITTA, LUNGHEZZA, citta, posizioniMax);
        break;
    case 3:
        //Visualizza tutti i nomi delle città con il corrispondente
        //valore di PM10
        visualizzaDati(NUM_CITTA, LUNGHEZZA, citta, particolato);
        break;
    }
    scelta = menu(opzioni, NUM_OPZIONI);
}
return 0;
}

```

menu.h

```

#pragma once
int menu(const char *opzioni, int numOpzioni);

```

menu.c

```

#include <stdio.h>
/**
 * Visualizza il menù e restituisce la scelta fatta dall'utente.
 * @param opzioni - Elenco di opzioni del menù
 * @param numOpzioni - Numero di opzioni presenti nel menù
 * @return la scelta dell'utente
 */
int menu(const char *opzioni, int numOpzioni) {
    int scelta;
    int inputOK = 0;

    printf("%s", opzioni);
    inputOK = scanf("%d", &scelta);
    while(getchar() != '\n');
    while(inputOK != 1 || (scelta < 1 || scelta > numOpzioni)) {
        printf("Errore di scelta\n");
        printf("%s", opzioni);
        inputOK = scanf("%d", &scelta);
        while(getchar() != '\n');
    }
    return scelta;
}

```

citta.h

```

#pragma once
void caricaDati(int numCitta, int lunghezza, char citta[][lunghezza],
float particolato[]);
int posizioneMassimo(int numCitta, float particolato[]);
void visualizzaDati(int numCitta, int lunghezza, char citta[][lunghezza],
float particolato[]);
void posizioniMassime(int numCitta, float particolato[], int posizioniMax[]);
void visualizzaMassimi(int numCitta, int lunghezza, char citta[][lunghezza],
int posizioni[]);

```

citta.c

```

#include <stdio.h>

```

```

/**

```

```
* Carica i nomi delle città e il corrispondente valore del PM10
* @param numCitta - Numero di città
* @param lunghezza - Lunghezza massima del nome di tutte le città
* @param citta[][] - Nomi delle città
* @param particolato[] - Valori di PM10 per ogni città
*/
void caricaDati(int numCitta,
               int lunghezza,
               char citta[][lunghezza],
               float particolato[]) {

    int inputOK; //Serve per verificare l'inserimento di un corretto
                //valore numerico
    //Ciclo di caricamento dei nomi delle città e del PM10
    //verificando che quest'ultimo sia corretto
    for(int i = 0; i < numCitta; ++i) {
        printf("Citta': ");
        scanf("%s", citta[i]);
        printf("PM10: ");
        inputOK = scanf("%f", &particolato[i]);
        while(getchar() != '\n');
        while(inputOK != 1 || particolato[i] < 0) {
            printf("Errore particolato\n");
            printf("PM10: ");
            inputOK = scanf("%f", &particolato[i]);
            while(getchar() != '\n');
        }
    }
}

/**
 * Individuazione della posizione del massimo.
 * @param numCitta - Numero di città
 * @param particolato[] - Valori di PM10 delle singole città
 * @return la posizione nell'array del primo valore massimo di
 *         PM10 trovato nell'array
 */
int posizioneMassimo(int numCitta, float particolato[]) {
    int posMax = 0; //Posizione del primo valore massimo di PM10
                //trovato nell'array
    float max = particolato[0]; //Valore massimo di PM10 presente
                //nell'array
    //Ciclo per individuare la posizione del valore massimo di PM10
    for(int i = 1; i < numCitta; ++i) {
        if(particolato[i] > max) {
            max = particolato[i];
            posMax = i;
        }
    }
    //Restituzione della posizione del valore massimo del PM10
    return posMax;
}

/**
 * Visualizzazione dell'elenco delle città con il relativo
```

```
* valore di PM10.
* @param numCitta - Numero di città
* @param lunghezza - Lunghezza massima del nome di tutte le città
* @param citta[][] - Nomi delle città
* @param particolato[] - Valori di PM10 per ogni città
*/
void visualizzaDati(int numCitta,
                   int lunghezza,
                   char citta[][lunghezza],
                   float particolato[]) {
    //Ciclo di visualizzazione delle città e del corrispondente PM10
    for(int i = 0; i < numCitta; ++i) {
        printf("%s: %f\n", citta[i], particolato[i]);
    }
}

/**
 * Individuazione di tutte le posizioni che contengono un
 * valore massimo di PM10.
 * @param numCitta - Numero di città
 * @param particolato[] - Valori di PM10 per ogni città
 * @param posizioniMax[] - Indici delle posizioni che corrispondono
 * ad un valore di PM10 pari al massimo
 */
void posizioniMassime(int numCitta,
                      float particolato[],
                      int posizioniMax[]) {
    //Ciclo di inizializzazione dell'array degli indici
    //delle posizioni dei massimi. Inizializzate a -1
    //in modo da avere un valore impossibile per un indice
    //da usare per terminare i futuri cicli senza dover
    //scorrere l'intero array
    for(int i = 0; i < numCitta; ++i) {
        posizioniMax[i] = -1;
    }
    //Individuazione della posizione del primo valore massimo di PM10
    int posMax = posizioneMassimo(numCitta, particolato);
    //Memorizzazione del valore massimo di PM10 da usare per i confronti
    float max = particolato[posMax];
    //Ciclo di individuazione e salvataggio delle posizioni dei valori
    //pari al massimo di PM10
    for(int i = 0, j = 0; i < numCitta; ++i) {
        if(particolato[i] == max) {
            //In questa istruzione bisogna usare necessariamente j++
            //e non ++j perché in questo modo prima viene usato il
            //valore di j per indicizzare la posizione in posizioniMax
            //e dopo viene incrementato il valore di j.
            //Se si usasse ++j prima verrebbe incrementato il valore di j
            //e poi questo sarebbe usato per indicizzare la posizione in
            //posizioniMax.
            posizioniMax[j++] = i;
        }
    }
}
```

```
/**
 * Visualizzazione di tutte le città che hanno un PM10 pari al massimo.
 * @param numCitta - Numero di città
 * @param lunghezza - Lunghezza massima del nome di tutte le città
 * @param citta[][] - Nomi delle città
 * @param posizioni[] - Indici delle posizioni che corrispondono
 *                      ad un valore di PM10 pari al massimo
 */
void visualizzaMassimi(int numCitta,
                      int lunghezza,
                      char citta[][lunghezza],
                      int posizioni[]) {
    //Ciclo di visualizzazione
    for(int i = 0; i < numCitta; ++i) {
        //Appena viene trovato nell'array posizioni[] un valore pari a -1
        //si è certi che non sono presenti altri indici validi
        if(posizioni[i] == -1) {
            break;
        } else {
            //Viene visualizzata la città che si trova all'indice indicato
            //dalla i-esima posizione di posizioni[]
            int pos = posizioni[i];
            printf("La città con il PM10 massimo a' %s\n", citta[pos]);
        }
    }
}
```

Array numerici e di stringhe

```
#include <stdio.h>
#include <stdlib.h>

#define RIGHE 3
#define COLONNE 3
#define LUN_STR 50

int main() {
    int voti[RIGHE][COLONNE];
    char nomi[COLONNE][LUN_STR];

    for(int i = 0; i < RIGHE; ++i) {
        printf("Inserire il nome: ");
        scanf("%s", nomi[i]);
        printf("Inserire i voti di %s\n", nomi[i]);
        for(int j = 0; j < COLONNE; ++j) {
            printf("Voto %d: ", j+1);
            scanf("%d", &voti[i][j]);
        }
    }
    return 0;
}
```

Uso di fgets()

```
#include <stdio.h>

#define DIM 5
#define NUM 3
#define LEN 50

int main(void) {
    //int eta[DIM];
    //char caratteri[DIM];
    //char nome[LEN];
    char nomi[NUM][LEN];

    for (int i = 0; i < NUM; i++) {
        printf("Inserisci il nome in posizione %d: ", i+1);
        fgets(nomi[i], sizeof(nomi[i]), stdin);
        int posizioneInvio = strcspn(nomi[i], "\n");
        nomi[i][posizioneInvio] = '\0';
    }

    for (int i = 0; i < NUM; i++) {
        printf("%s", nomi[i]);
    }

    return 0;
}
```

Esercizi

107. Progettare un algoritmo che permetta di indicare se uno studente è sufficiente o meno di una materia basandosi sulla media dei voti della materia. Visualizzare la media e l'elenco dei voti dopo aver visualizzato la media. Caricare i voti della materia in un array.

108. Progettare un algoritmo che determini l'altezza minima e l'altezza massima degli studenti di una classe fornita in input, indicando anche quanti studenti hanno un'altezza pari al minimo e quanti hanno un'altezza pari al massimo. Si visualizzino anche tutte le altezze inserite con gli scarti rispetto all'altezza minima e all'altezza massima. Usare un menù per gestire ogni singolo aspetto dell'applicazione rispettando i vincoli descritti di seguito. Il menù può essere il seguente:

- 1) Inserisci la classe
 - 2) Carica altezze
 - 3) Visualizza altezza minima
 - 4) Visualizza altezza massima
 - 5) Visualizza altezze con scarto dal minimo e dal massimo
 - 6) Esci
- Scelta: _

VINCOLI

- a. Se non è stata eseguita la scelta (1) gli altri punti non possono essere eseguiti.
- b. Se non è stata eseguita la scelta (2) i punti (3), (4) e (5) non possono essere eseguiti.
- c. Se non sono state eseguite le scelte (3) e (4) il punto (5) non può essere eseguito.

109. Progettare un algoritmo che permetta di indicare se uno studente è sufficiente o meno di una materia basandosi sulla media dei voti della materia. Visualizzare la media e l'elenco dei voti dopo aver visualizzato la media. Caricare i voti della materia in un array. Usare un menù per gestire ogni singolo aspetto dell'applicazione rispettando i vincoli descritti di seguito. Il menù può essere il seguente:

- 1) Inserisci studente
 - 2) Carica voti
 - 3) Visualizza voti
 - 4) Visualizza stato studente
 - 5) Esci
- Scelta: _

VINCOLI

- a. Se non è stata eseguita la scelta (1) gli altri punti non possono essere eseguiti.
- b. Se non è stata eseguita la scelta (2) i punti (3) e (4) non possono essere eseguiti.

110. Progettare un algoritmo che permetta di indicare se gli studenti di una classe sono sufficienti o meno in una determinata materia, basandosi sulla media dei voti della materia. Per ogni studente visualizzare la media e l'elenco dei voti dopo aver visualizzato la media. Caricare i nomi degli studenti in un array di stringhe. Caricare i voti della materia per ogni studente in un array numerico. I due array devono essere gestiti in modo parallelo. Usare un menù per gestire ogni singolo aspetto

dell'applicazione rispettando i vincoli descritti di seguito. Il menù può essere il seguente:

- 1) Inserisci gli studenti
- 2) Carica voti degli studenti
- 3) Visualizza voti di ogni studente
- 4) Visualizza stato di ogni studente
- 5) Esci

Scelta: _

VINCOLI

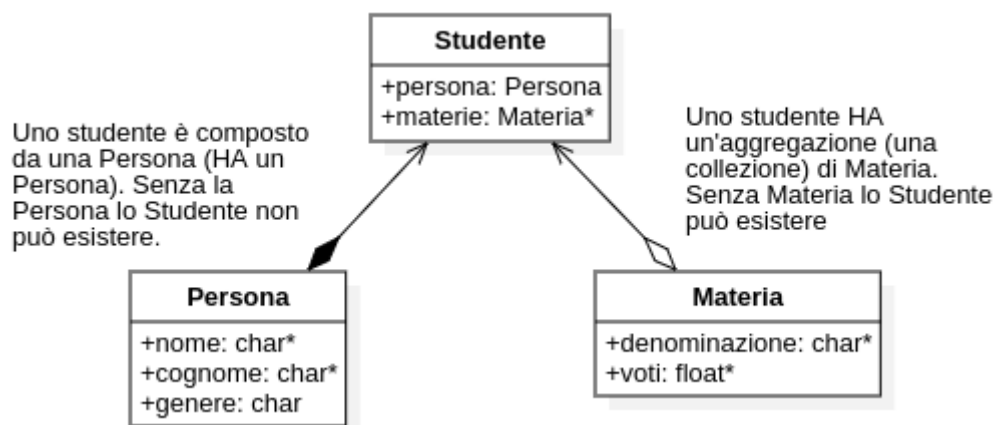
- a. Se non è stata eseguita la scelta (1) gli altri punti non possono essere eseguiti.
- b. Se non è stata eseguita la scelta (2) i punti (3) e (4) non possono essere eseguiti.

Per chi vuole ampliare gli orizzonti e conosce già le struct

Provare a sviluppare le seguenti versioni:

- I. Creare una struct `Studente` con due campi: il nominativo dello studente e l'array dei voti della materia. Usare la struct `Studente` per creare una array di studenti.
- II. Creare una struct `Persona` che contenga: nome, cognome, genere. La struct `Persona` deve sostituire il campo nominativo nella struct `Studente`. Usare la struct `Studente` per creare una array di studenti.
- III. Creare una struct `Materia` che contenga: la denominazione della materia e l'array dei voti della materia. Usare la struct `Materia` per sostituire i voti nella struct `Studente`. In questo modo per ogni studente si hanno i voti di tutte le materie. La struct `Studente` deve quindi avere un campo di tipo struct `Persona` e un campo array di tipo struct `Materia`.

Di seguito viene mostrato un possibile diagramma delle struct (classi) per l'estensione III.



111. Progettare un'applicazione che permette di gestire le medie finali di una classe. Della classe si vogliono caricare i nomi degli studenti e la media finale per ogni studente. Una volta caricati i dati l'applicazione deve fornire la possibilità di visualizzare l'elenco di tutti gli studenti con le relative medie. Inoltre si vuole ottenere a video l'elenco degli studenti con la media migliore e l'elenco di quelli con la media peggiore. Si usi un menù per gestire le diverse opzioni richieste. Si faccia riferimento all'esempio [Array paralleli stringhe con elenco di singoli numeri](#) per capire come gestirli.

112. Progettare un'applicazione che permetta di inserire i nomi e le età dei tuoi compagni di classe in due array paralleli. L'applicazione deve permettere di inserire un'età di riferimento per separare gli studenti con un'età pari o inferiore a questa, da quelli con un'età superiore. Per fare questa separazione si suggerisce di usare altri quattro vettori. Due vettori paralleli devono contenere nomi ed età degli studenti con età pari o inferiore a quella di riferimento. Gli altri due vettori devono contenere i nomi e l'età degli studenti con età superiore a quella di riferimento. Usare un menù per gestire ogni singolo aspetto dell'applicazione rispettando i vincoli descritti di seguito. Il menù può essere il seguente:

- 1) Inserisci gli studenti e le loro età
 - 2) Carica l'età di riferimento
 - 3) Visualizza gli studenti e l'età $\leq$ al riferimento
 - 4) Visualizza gli studenti e l'età $>$ al riferimento
 - 5) Esci
- Scelta: _

VINCOLI

- c. Se non è stata eseguita la scelta (1) gli altri punti non possono essere eseguiti.
- d. Se non è stata eseguita la scelta (2) i punti (3) e (4) non possono essere eseguiti.

113. Progettare un algoritmo che implementi il gioco dell'impiccato. Si preveda un dizionario di parole precedentemente salvato in una struttura dati. L'algoritmo dovrà selezionare in modo pseudo-casuale una parola dal dizionario e usarla per il gioco. Per ogni tentativo fallito si visualizzi una frase che indichi quali sono le parti del corpo già perdute (testa, corpo, braccio destro, braccio sinistro, gamba destra, gamba sinistra). Per ogni tentativo dovrà essere visualizzata la sequenza di caratteri indovinata nella giusta posizione e tutta la sequenza di caratteri usata fino a quel momento. L'utente dovrà inserire un carattere per volta come tentativo per completare la parola da indovinare. Si utilizzi la programmazione modulare per implementare il gioco. Si crei l'analisi e l'applicazione.

114. In un gioco il proprio avatar deve impossessarsi di due poteri particolari, il potere dell'aria e il potere dell'acqua. Il primo è custodito nella vetta più alta del territorio, mentre il secondo nell'abisso più profondo. Si progetti un algoritmo che per prima cosa generi in modo casuale il territorio di gioco con le misure delle altitudini per ogni zona e, successivamente simuli il percorso di ricerca dell'avatar che va alla ricerca dei due poteri percorrendo il territorio. Il territorio può essere percorso in diversi modi, partendo da uno dei quattro punti cardinali. Il giocatore deve scegliere da un menù il punto di partenza ed avviare così la ricerca dell'avatar che deve percorrere tutto il territorio alla ricerca dei poteri. Il gioco termina dopo che l'avatar ha trovato entrambi i poteri. Per simulare il tempo di ricerca si supponga che l'avatar impieghi un secondo per percorrere ogni casella del territorio. Al termine del gioco deve essere visualizzato il tempo impiegato per trovare i due poteri. Per svolgere l'esercizio si dovranno usare le [matrici](#) e la programmazione modulare. Per simulare il tempo di percorrenza di una zona del territorio usa la funzione [sleep\(\)](#). Si crei l'analisi e l'applicazione.

115. Progettare un'applicazione che permetta di inserire una stringa eliminando il ritorno a capo ('\n') e poi faccia scegliere all'utente, attraverso un menù, una delle seguenti

operazioni di manipolazione da effettuare sulla stringa. La stringa deve essere inserita usando la funzione `fgets()`.

- 1) Reverse → Visualizza la stringa al contrario
 - 2) Count → Conta il numero di caratteri effettivamente presenti nella stringa
 - 3) Vowels → Conta il numero di vocali presenti nella stringa
 - 4) Binary Inversion → Inverte i caratteri in posizione pari con quelli in posizione dispari
 - 5) SubstituteChar → Sostituisce un carattere selezionato della stringa con uno fornito da tastiera.
 - 6) SubstitutePos → Sostituisce un carattere della stringa in una determinata posizione con uno fornito da tastiera.
- Si crei l'analisi e l'applicazione.

116. Progettare un algoritmo che, dato in input un numero intero positivo espresso in base 10, fornisca la sua conversione in base 2 (si utilizzino gli array).
117. Progettare un algoritmo che converta un numero binario in un numero in base 10. Il numero binario è rappresentato su N bit, e il valore di N dovrà essere fornito dall'utente. L'utente dovrà inserire le cifre del numero binario un bit alla volta, partendo dal bit meno significativo (ossia dal bit di peso 2^0). Il programma visualizza il numero in base 10 corrispondente (si utilizzino gli array).
118. Progettare un algoritmo che permetta di indicare se uno studente è ammesso o non ammesso alla classe successive in base alla media finale. La media finale dovrà essere calcolata come media delle medie di tutte le materie e, se risulterà ≥ 6 allora lo studente sarà ammesso alla classe successiva, altrimenti lo studente non sarà ammesso alla classe successiva.
119. Progettare un algoritmo che acquisisca da tastiera un vettore di interi di dimensione N e trovi il minimo, il massimo e la media degli elementi.
120. Dato un vettore di valori numerici, progettare un algoritmo che conti quanti elementi di un array sono compresi tra un valore minimo e un valore massimo forniti da tastiera, visualizzando sia il conteggio, sia l'elenco dei valori numerici.
121. Progettare un algoritmo che permetta di leggere 10 numeri interi e visualizzi la sequenza memorizzata senza le eventuali ripetizioni. Ad esempio, se nel vettore fossero memorizzati i valori: 15, 3, 5, 3, 11, 5, 15, 5, 15, 11, l'algoritmo dovrà visualizzare i valori 15, 3, 5, 11.
122. Progettare un algoritmo che permetta di inserire N valori interi in un vettore e, dato un valore K, generi altri due vettori contenenti rispettivamente tutti i valori non maggiori di K e tutti i valori maggiori di K.
123. Progettare un algoritmo che permetta di inserire le età dei tuoi compagni di classe in un array e, data un'età di riferimento, generi altri due vettori contenenti rispettivamente tutte le età non maggiori dell'età di riferimento e tutte le età maggiori dell'età di riferimento.
124. Progettare un algoritmo che inverta la posizione degli elementi di un array di interi di cui sia fornita la dimensione (risolvere il problema senza usare vettori ulteriori).

125. Progettare un algoritmo che memorizzi in un array le ore di studio di uno studente per ogni giorno del mese. L'algoritmo deve calcolare e visualizzare il numero totale di ore passate a studiare nel corso del mese, il numero di giorni in cui lo studente ha studiato per più ore e il numero di giorni in cui lo studente ha studiato di meno.
126. In un array di 9 elementi sono memorizzate le presenze mensili degli studenti di una classe durante l'anno scolastico. Progettare un algoritmo che determini:
- la media mensile di presenze nel corso dell'intero anno scolastico;
 - il numero totale di presenze nel primo quadrimestre;
 - il mese in cui si è registrato il numero massimo di presenze;
 - il mese in cui si è registrato il numero minimo;
 - la media delle presenze del secondo quadrimestre.
127. Progettare un algoritmo che permetta di indovinare in 6 tentativi un codice numerico composto da sei cifre nell'intervallo [100000 - 999999]. Il codice numerico deve essere generato come numero pseudo casuale e deve essere salvato in un array di interi, cifra per cifra. Per la generazione del numero pseudo casuale si utilizzino le funzioni `time()` per recuperare il tempo corrente del sistema, `srand()` per impostare il "seme", cioè il punto di inizio per la generazione del numero pseudo casuale, e `rand()` per la generazione effettiva del numero pseudo casuale in un dato intervallo. Di seguito viene fornito un esempio per la generazione di un numero pseudo casuale:

```
srand(time(0)); // Imposta il tempo corrente del sistema come seme per la
                // generazione del numero pseudo casuale
int num = (rand() % (UPPER - LOWER + 1)) + LOWER;
// Genera un numero pseudo casuale nell'intervallo [LOWER - UPPER). La
funzione rand() genera un numero pseudo casuale nell'intervallo [0 - 1),
l'uso dei valori LOWER e UPPER permettono di variare l'intervallo di
generazione
```

128. Uno strumento di misura fornisce un dato ogni minuto nell'arco di un'ora. Per ovviare a possibili errori si vogliono elaborare i valori rilevati sostituendoli con una media a tre punti: ogni elemento viene sostituito dalla media di sé stesso, dell'elemento che lo precede e quello che lo segue. Per i due elementi estremi viene considerato due volte il valore dell'elemento stesso e una volta il successivo o il precedente nel caso si tratti rispettivamente del primo o dell'ultimo elemento. Progettare un algoritmo in grado di visualizzare i valori finali.
129. Scrivere un algoritmo che restituisca il valore massimo di una matrice di dimensioni $N \times N$ fornita, indicando anche qual è la sua posizione nella matrice (numero riga, numero colonna). Per svolgere l'esercizio si dovranno usare le [matrici](#).
130. Progettare un algoritmo che preveda il caricamento dei nomi e delle età degli studenti della tua classe e visualizzi le seguenti informazioni:
- elenco dei nomi e delle età degli studenti maggiorenni;
 - elenco dei nomi e delle età degli studenti più vecchi;
 - elenco dei nomi e delle età degli studenti più giovani;
 - media delle età;
 - elenco dei nomi e le età degli studenti che hanno un'età maggiore della media;

f. elenco dei nomi e le età che hanno un'età minore della media.

131. Progettare un algoritmo che preveda il caricamento dei cognomi e delle età degli studenti della tua classe e visualizzi prima l'elenco dei cognomi degli studenti maggiorenni e poi l'elenco dei cognomi degli studenti minorenni.

132. Progettare un algoritmo che determini in un gruppo di persone quali risultano essere sottopeso, quali normopeso, quali in sovrappeso e quali obese. Visualizzare gli elenchi delle persone che ricadono in ognuno dei gruppi.

Si consideri che l'Indice di Massa Corporea (IMC) di una persona si determina con la seguente formula:

$$\text{IMC} = \text{peso (kg)} / \text{altezza}^2 (\text{m}^2)$$

e che l'Organizzazione Mondiale della Sanità individua le seguenti quattro categorie:

- a. sottopeso: IMC inferiore a 19;
- b. normopeso: IMC nell'intervallo [19, 24];
- c. sovrappeso: IMC nell'intervallo (24, 30];
- d. obeso: IMC superiore a 30.

133. Progettare un algoritmo che visualizzi la scritta "Sì, contiene lettere estranee all'alfabeto italiano" se una parola data in input contiene una lettera estranea all'alfabeto italiano (j, k, x, y, w), altrimenti visualizza la stringa "NO, non contiene lettere estranee all'alfabeto italiano".

134. Progettare un algoritmo che preveda l'inserimento di una stringa e di un carattere e visualizzi il numero di occorrenze del carattere nella stringa.

135. Progettare un algoritmo che preveda l'inserimento di N stringhe e di un singolo carattere e visualizzi il numero massimo di occorrenze del carattere rispetto a tutte le N scritte.

Ad esempio se il carattere da cercare è 'a' e le stringhe sono 2, ad esempio "aaabbcc" e "bbaaaaacc", il valore da visualizzare sarà 5 perché nella prima stringa il carattere 'a' occorre 3 volte, mentre nella seconda stringa il carattere 'a' occorre 5 volte e quindi il massimo numero di occorrenze è presente nella seconda stringa.

136. Progettare un algoritmo che visualizzi la tabella ASCII standard con il seguente formato partendo dal carattere *Space*:

num. decimale - num. esadecimale - carattere

Ad esempio:

[link](#)

137. Progettare un algoritmo che, dato un elenco di parole scritte in minuscolo, le converta tutte in maiuscolo, senza usare la funzione `toupper()`.

138. Progettare un algoritmo che permetta di inserire tre parole e le ordini lessicograficamente. Ad esempio, inserendo le parole Caterina, Abele, Barbara, dovranno essere visualizzate come

Able

Barbara

Caterina

139. Progettare un algoritmo che legga una stringa, costituita da caratteri (maiuscole, minuscole, à, è, é, ì, ò, ù) e da simboli di punteggiatura. L'algoritmo dovrà analizzare la stringa e visualizzare le seguenti informazioni:
- a. per ognuna delle lettere dell'alfabeto, il numero di volte che la lettera compare nella stringa;
 - b. il numero di consonanti presenti nella stringa;
 - c. il numero di vocali presenti nella stringa;
 - d. il numero di simboli di punteggiatura.

ALTERNATIVA semplice - Progettare un algoritmo che legga un array di interi e lo analizzi visualizzando le seguenti informazioni:

- a. il numero di valori positivi;
- b. il numero di valori negativi;
- c. il numero di valori pari a zero.

ALTERNATIVA avanzata - Progettare l'algoritmo originale creando una funzione di ricerca che permetta di cercare qualsiasi sequenza di caratteri all'interno della stringa originale.

140. Progettare un algoritmo che implementi il gioco dell'impiccato. Si preveda un dizionario di parole precedentemente salvato in una struttura dati. L'algoritmo dovrà selezionare in modo pseudo-casuale una parola dal dizionario e usarla per il gioco. Per ogni tentativo fallito si visualizzi una frase che indichi quali sono le parti del corpo già perdute (testa, corpo, braccio destro, braccio sinistro, gamba destra, gamba sinistra). Per ogni tentativo dovrà essere visualizzata la sequenza di caratteri indovinata nella giusta posizione e tutta la sequenza di caratteri usata fino a quel momento. L'utente dovrà inserire un carattere per volta come tentativo per completare la parola da indovinare.
141. Progettare un algoritmo che permetta di visualizzare le seguenti informazioni degli stati appartenenti alla Comunità Europea, permettendo di scegliere tramite un menù quale punto visualizzare. Si scelgano opportunamente le strutture dati per salvare i dati necessari e si ricerchino questi ultimi su Internet per popolarle.
- a. Lo stato con la popolazione maggiore.
 - b. Lo stato con la popolazione minore.
 - c. L'elenco degli stati e delle relative popolazioni in ordine crescente di popolazione.
 - d. L'elenco degli stati e delle relative popolazioni in ordine decrescente di nome di stato.

Produrre tutta l'analisi (macroanalisi, microanalisi, tabella delle variabili) e il codice commentato opportunamente in ogni sua parte. Il progetto si presta molto bene all'uso delle funzioni che per ora potrete scegliere se usare o meno; nel caso usaste le funzioni la microanalisi e la tabella delle variabili dovrà essere svolta per ogni singola funzione separatamente.

142. Progettare un algoritmo che verifichi se le parole inserite in input, contenenti la sequenza «QU» al loro interno, abbiano sempre la Q seguita da una U (es. riquadro, quaderno). L'algoritmo dovrà visualizzare la parola e la stringa Sì nel caso che la sequenza «QU» sia corretta (es. riquadro Sì), visualizzerà invece la parola e la stringa No nel caso la Q non sia seguita dalla U (es. qweit No), mentre visualizzerà solo la parola se questa non contiene il carattere Q (es. pirata).

143. Progettare un algoritmo che, a partire da una parola inserita in input, conti quante volte è presente la sequenza di lettere «QU» senza prevedere un limite superiore per il numero di sequenze presenti nella parola stessa.

Struct

C Tutorial

15.C Programming Arrays by [Programiz](#)

- a. [C struct](#)
- b. [C structs and Pointers](#)
- c. [C Structure and Function](#)

16.VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)

- a. [struct Basics](#) ← **IMPORTANTE!!!** *Per ora puoi fermarti al minuto 17:39, la funzione malloc() la vedremo più avanti*

17.Il seguente link è un articolo che tratta la gestione avanzata delle struct. Potrebbe essere difficile affrontarlo ora, ma puoi riprovare a leggerlo fra un po' di tempo.

[How to struct - lessons on Structures in C](#) by [Orunmila](#), hosted by [microforum.cc](#) ← *facoltativo*

Struct e programmazione modulare

Strutture esplicite

L'utilizzo della programmazione modulare permette di gestire un'applicazione in modo tale che le modifiche di un modulo non vadano ad influenzare il resto del programma. È quindi importante utilizzare questa tecnica di programmazione anche con le struct.

Il primo passo per rendere modulare un'applicazione che usa le struct è quello di definire la struttura in un header separato, incluso nei moduli in cui viene usata la struttura. L'obiettivo futuro sarà quello di confinare completamente la struttura all'interno del file .c che contiene la definizione completa delle funzioni che la manipolano, al fine di applicare il principio di **information hiding** e di **incapsulamento**. Per ora però le struct vengono usate in modo esplicito e, per questa ragione, il compilatore ha necessità di conoscere completamente la dimensione esatta della struttura per allocare lo spazio sullo stack. Per questo la sua definizione avviene completamente in un file header separato.

Esempio di programmazione modulare con strutture esplicite - Persona

L'esempio tratta la gestione dei dati di una persona. Della persona si vogliono memorizzare un identificatore, il nome e il cognome. Per gestire tutti i dati in modo unitario viene creata una **struct Persona** che viene usata come nuovo tipo per la variabile manipolata nel **main()**.

Viene previsto un header di definizioni per definire la struct e le macro necessarie. Questo header viene incluso in ogni modulo che necessita di manipolare la struttura.

È presente anche il modulo persona.c che contiene la definizione delle funzioni che manipolano la struttura. In questo modulo si iniziano a creare le prime funzioni di manipolazione di una variabile del tipo della struttura, partendo dalla funzione per creare un nuovo oggetto di tipo Persona

```
struct Persona newPersona(int id, char *nominativo, char *indirizzo)
```

e la funzione per modificare l'indirizzo

```
void setIndirizzo(struct Persona *persona, const char *indirizzo)
```

Non si prevede la modifica dell'id e del nominativo.

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include "definizioni.h"
#include "persona.h"

int main(void) {
    struct Persona miaPersona;
    int id;
    char nominativo[LEN_NOMINATIVO];
```

```
char indirizzo[LEN_INDIRIZZO];

// Caricamento dati per creare una nuova persona
printf("Id: ");
scanf("%d", &id);
getchar();
printf("Nominativo: ");
fgets(nominativo, LEN_NOMINATIVO, stdin);
nominativo[strlen(nominativo) - 1] = '\0';
printf("Indirizzo: ");
fgets(indirizzo, LEN_INDIRIZZO, stdin);
indirizzo[strlen(indirizzo) - 1] = '\0';

// Creazione di una nuova persona
miaPersona = newPersona(id, nominativo, indirizzo);

// Visualizzazione dei dati della persona
printf("%d - %s, %s\n", miaPersona.id, miaPersona.nominativo,
        miaPersona.indirizzo);

// Modifica dell'indirizzo della persona
bool flag = setIndirizzo(&miaPersona, "Via Roma, 1");
if (!flag) printf("Indirizzo non modificato\n");
flag = setIndirizzo(&miaPersona, "");
if (!flag) printf("Indirizzo non modificato\n");

// Visualizzazione dei dati della persona modificati
printf("%d - %s, %s\n", miaPersona.id, miaPersona.nominativo,
        miaPersona.indirizzo);

return 0;
}

definizioni.h
#ifndef DEFINIZIONI_H
#define DEFINIZIONI_H

#define LEN_NOMINATIVO 20
#define LEN_INDIRIZZO 20

// Dichiarazione completa della struttura per permettere l'uso diretto
struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
};

#endif // DEFINIZIONI_H
```

persona.h

```
#pragma once
// Crea una nuova persona
struct Persona newPersona(int id, char *nominativo, char *indirizzo);
// Modifica l'indirizzo di una persona
bool setIndirizzo(struct Persona *persona, const char *indirizzo);
```

persona.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include "definizioni.h"

struct Persona newPersona(int id, char *nominativo, char *indirizzo) {
    struct Persona persona;

    persona.id = id;
    strcpy(persona.nominativo, nominativo);
    strcpy(persona.indirizzo, indirizzo);
    return persona;
}

bool setIndirizzo(struct Persona *persona, const char *indirizzo) {
    if (strcmp(indirizzo, "") != 0) {
        strcpy(persona->indirizzo, indirizzo);
        return true;
    }
    return false;
}
```

Una possibile esecuzione produce il seguente risultato.

```
Id: 1
Nominativo: Minni
Indirizzo: Via Torino, 6
1 - Minni, Via Torino, 6
1 - Minni, Via Roma, 1
```

Variabili static

C Tutorial

18.VIDEO by [Portfolio Courses](#) in [C Programming Tutorial](#)

- o. [Static Local Variables](#). Cosa hai scoperto di nuovo in questo video oltre all'uso del modificatore static? Guarda il video (a) sui [file](#).
- p. [Static Global Variables](#) → *Approfondimento facoltativo*
- q. [Static Functions](#) → *Approfondimento facoltativo*

Esempio con strutture esplicite e modificatore static - Array di persone

La seguente applicazione usa la struttura Persona vista precedentemente e ne migliora il costruttore imponendo la generazione automatica dell'id di ogni persona usando una variabile static.

Inoltre gestisce una collezione di studenti che contiene elementi di tipo **struct Persona**. L'applicazione permette di caricare l'array, visualizzarne il contenuto in modo ordinato per nominativo e, cercare un nominativo per farsi restituire tutti i dati.

main.c

```
/**
 * L'applicazione carica, ordina e visualizza una collezione di studenti
 */
#include <stdio.h>
#include <string.h>
#include "lib/definizioni/definizioni.h"
#include "lib/persona/persona.h"
#include "lib/arraypersona/arraypersona.h"

int main() {
    int numStudenti;

    // Caricamento del numero di studenti
    numStudenti = numeroElementi("Numero studenti: ", "Numero studenti errato",
1, 30);
    // Dichiarazione dell'array di studenti
    struct Persona studenti[numStudenti];
    // Caricamento degli studenti
    caricaArray(studenti, numStudenti, "Inserisci i dati dello studente");
    // Visualizzazione degli studenti
    visualizzaArray(studenti, numStudenti);
    // Ordinamento degli studenti in base al nominativo
    ordinaArray(studenti, numStudenti);
    // Visualizzazione degli studenti ordinati
    visualizzaArray(studenti, numStudenti);
    // Ricerca di uno studente in base al nominativo
    char nominativo[LEN_NOMINATIVO];
    printf("Inserisci il nominativo dello studente da cercare: ");
    fgets(nominativo, LEN_NOMINATIVO, stdin);
```

```
nominativo[strlen(nominativo) - 1] = '\\0';
int pos = ricercaElemento(studenti, numStudenti, nominativo);
if (pos == -1) {
    printf("Studente non trovato.\\n");
} else {
    printf("Studente trovato:\\n");
    printf("%d - %s - %s\\n", studenti[pos].id, studenti[pos].nominativo,
        studenti[pos].indirizzo);
}

return 0;
}
```

definizioni.h

```
#ifndef DEFINIZIONI_H
#define DEFINIZIONI_H

#define LEN_NOMINATIVO 20
#define LEN_INDIRIZZO 20
#define NUM_STUDENTI 3

// Dichiarazione completa della struttura per permettere l'uso diretto
struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
};

#endif // DEFINIZIONI_H
```

persona.h

```
#pragma once
#include "../definizioni/definizioni.h"

// Crea una nuova persona
struct Persona newPersona(char *nominativo, char *indirizzo);
// Imposta l'indirizzo della persona specificata
bool setIndirizzo(struct Persona *persona, const char *indirizzo);
```

persona.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include "../definizioni/definizioni.h"

/**
 * Creazione di una nuova persona alla quale viene assegnato
 * un id univoco
 * @param nominativo - Nominativo della persona
 * @param indirizzo - Indirizzo della persona
```

```
* @return la nuova persona creata
*/
struct Persona newPersona(char *nominativo, char *indirizzo) {
    struct Persona persona;
    // Permette di assegnare un id unico ad ogni persona creata
    static int indice = 0;

    persona.id = ++indice;
    strcpy(persona.nominativo, nominativo);
    strcpy(persona.indirizzo, indirizzo);
    return persona;
}

/**
 * Imposta l'indirizzo di una persona
 * @param persona - Persona a cui cambiare l'indirizzo
 * @param indirizzo - Nuovo indirizzo della persona
 */
bool setIndirizzo(struct Persona *persona, const char *indirizzo) {
    if (strcmp(indirizzo, "") != 0) {
        strcpy(persona->indirizzo, indirizzo);
        return true;
    }
    return false;
}
```

arraypersona.h

```
#pragma once
#include "../definizioni/definizioni.h"

// Input del numero di elementi
int numeroElementi(const char *msg, const char *msgErrore, int minore, int maggiore);
// Caricamento array di elementi
void caricaArray(struct Persona array[], int numElementi, const char *msg);
// Visualizzazione del contenuto dell'array
void visualizzaArray(struct Persona array[], int numElementi);
// Ordinamento lessicografico dell'array
void ordinaArray(struct Persona array[], int numElementi);
// Ricerca di un elemento dell'array
int ricercaElemento(struct Persona array[], int numElementi, char *criterio);
```

arraypersona.c

```
#include <stdio.h>
#include <string.h>
#include "../definizioni/definizioni.h"
#include "../persona/persona.h"
```

```
/**
```

```
* Input del numero di elementi
* @param msg - Messaggio di richiesta dell'input
* @param msgErrore - Messaggio di segnalazione di errore
* @param minore - Estremo minore del dominio di validità
* @param maggiore - Estremo superiore del dominio di validità
* @return numero di elementi
*/
int numeroElementi(const char *msg, const char *msgErrore, int minore, int
maggiore) {
    int numElementi;
    int inputOK = 0;

    printf("%s", msg);
    inputOK = scanf("%d", &numElementi);
    while(getchar() != '\n');
    while (inputOK != 1 || numElementi < minore || numElementi > maggiore) {
        printf("%s\n", msgErrore);
        printf("%s", msg);
        inputOK = scanf("%d", &numElementi);
        while(getchar() != '\n');
    }
    return numElementi;
}

/**
* Carica l'array elementi.
* @param array - Array di elementi
* @param numElementi - Numero di elementi da caricare nell'array
* @param msg - Messaggio di richiesta dell'input
*/
void caricaArray(struct Persona array[], int numElementi, const char *msg) {
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
    // Caricamento degli elementi
    for (int i = 0; i < numElementi; i++) {
        // Messaggio di richiesta dell'input
        printf("%s %d:\n", msg, i + 1);
        // Caricamento dati per creare una nuova persona
        printf("Nominativo: ");
        fgets(nominativo, LEN_NOMINATIVO, stdin);
        int pos = strlen(nominativo) - 1;
        nominativo[pos] = '\0';
        printf("Indirizzo: ");
        fgets(indirizzo, LEN_INDIRIZZO, stdin);
        pos = strlen(indirizzo) - 1;
        indirizzo[pos] = '\0';
        // Creazione di una nuova persona
        array[i] = newPersona(nominativo, indirizzo);
    }
}
```

```
/**
 * Visualizza una linea decorativa parametrizzata
 * @param larghezzaNome - numero di "-" da visualizzare in base alla larghezza
 *                        del nome
 * @param larghezzaIndirizzo - numero di "-" da visualizzare in base alla
 *                        larghezza dell'indirizzo
 */
void stampaLinea(int larghezzaNome, int larghezzaIndirizzo) {
    printf("+----+");
    for (int i = 0; i < larghezzaNome; i++) printf("-");
    printf("+");
    for (int i = 0; i < larghezzaIndirizzo; i++) printf("-");
    printf("+\n");
}

/**
 * Visualizza elementi puntati dall'array.
 * @param array - Array di elementi
 * @param numElementi - Numero di elementi nell'array
 */
void visualizzaArray(struct Persona array[], int numElementi) {
    stampaLinea(LEN_NOMINATIVO, LEN_INDIRIZZO);
    printf("| Id |%-*s|%-*s|\n", LEN_NOMINATIVO, "Nome",
                                                LEN_INDIRIZZO, "Indirizzo");
    stampaLinea(LEN_NOMINATIVO, LEN_INDIRIZZO);

    for (int i = 0; i < numElementi; i++) {
//
https://stackoverflow.com/questions/1809399/how-to-format-strings-using-printf-
to-get-equal-length-in-the-output
        printf("|%4d|%-*s|%-*s|\n", array[i].id,
                                    LEN_NOMINATIVO, array[i].nominativo,
                                    LEN_INDIRIZZO, array[i].indirizzo);
        stampaLinea(LEN_NOMINATIVO, LEN_INDIRIZZO);
    }
}

/**
 * Ordina l'array di elementi seguendo un ordinamento lessicografico crescente.
 * @param array - Array da ordinare
 * @param numElementi - Numero di elementi presenti nell'array
 */
void ordinaArray(struct Persona array[], int numElementi) {
    for (int i = 0; i < numElementi - 1; i++) {
        for (int j = 0; j < numElementi - i - 1; j++) {
            if (strcmp(array[j].nominativo, array[j + 1].nominativo) > 0) {
                struct Persona temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```

```

    }
}
}
}

/**
 * Ricerca un elemento nell'array.
 * Viene usata la ricerca binaria e quindi l'array deve essere ordinato.
 * @param array - Array di elementi
 * @param numElementi - Numero di elementi presenti nell'array
 * @param criterio - Elemento da cercare
 * @return puntatore alla persona trovata o NULL se non trovata
 */
int ricercaElemento(struct Persona array[], int numElementi, char *criterio)
{
    // Imposto l'indice min e max ai valori iniziali
    int min = 0;
    int max = numElementi - 1;
    // Avvio il ciclo di ricerca binaria e ciclo fino a quando min <= max.
    while(min <= max) {
        // Imposto l'indice guess al valore medio tra min e max
        int guess = (min + max) / 2;
        // Se il nominativo della persona puntata da guess è uguale al criterio...
        if (strcmp(array[guess].nominativo, criterio) == 0) {
            // ...ritorno la posizione dell'array
            return guess;
        } else {
            // Se il nominativo della persona puntata da guess è minore del criterio...
            if (strcmp(array[guess].nominativo, criterio) < 0) {
                // ...imposto l'indice min al valore successivo a quello di guess
                min = guess + 1;
            } else {
                // altrimenti, se il nominativo della persona puntata da guess è maggiore
                // del criterio imposto l'indice max al valore precedente a quello di guess
                max = guess - 1;
            }
        }
    }
    // Se min diventa maggiore di max il nominativo non esiste e restituisco -1
    return -1;
}

```

Di seguito viene mostrata un'esecuzione di esempio.

Numero studenti: 5

Inserisci i dati dello studente 1:

Nominativo: Pippo Pippoli

Indirizzo: Via Roma, 6

Inserisci i dati dello studente 2:

Nominativo: Minni Minnoli

Indirizzo: Via Milano, 3

Inserisci i dati dello studente 3:

Nominativo: Paolino Paperino

Indirizzo: Via Torino, 5

Inserisci i dati dello studente 4:

Nominativo: Pluto Plutoli

Indirizzo: Via Saluzzo, 1

Inserisci i dati dello studente 5:

Nominativo: Clara Bella

Indirizzo: Via Sassari, 9

+-----+			+-----+		
Id	Nome		Indirizzo		
+-----+					
1	Pippo Pippoli		Via Roma, 6		
+-----+					
2	Minni Minnoli		Via Milano, 3		
+-----+					
3	Paolino Paperino		Via Torino, 5		
+-----+					
4	Pluto Plutoli		Via Saluzzo, 1		
+-----+					
5	Clara Bella		Via Sassari, 9		
+-----+					
+-----+					
Id	Nome		Indirizzo		
+-----+					
5	Clara Bella		Via Sassari, 9		
+-----+					
2	Minni Minnoli		Via Milano, 3		
+-----+					
3	Paolino Paperino		Via Torino, 5		
+-----+					
1	Pippo Pippoli		Via Roma, 6		
+-----+					
4	Pluto Plutoli		Via Saluzzo, 1		
+-----+					

Inserisci il nominativo dello studente da cercare: Pippo Pippoli

Studente trovato:

1 - Pippo Pippoli - Via Roma, 6

Esercizi

144. Ti viene chiesto di progettare la struttura dati per rappresentare un aeroplano. Dell'aereo si vogliono memorizzare diverse caratteristiche, come la marca, il modello, il numero dei motori, l'altitudine e la velocità. Crea un'applicazione che usi la nuova struttura che definisce l'aereo per caricare i dati relativi e, successivamente, visualizzarli.

Obiettivo - L'applicazione serve per testare il funzionamento di una struttura dati definita dal programmatore per gestire un aeroplano.

Progettazione strutture dati

Viene definito un nuovo tipo **struct** **Aereo** perché ...

```
#define LEN 30
```

```
struct Aereo {
    char marca[LEN];
    char modello[LEN];
    int numMotori;
    float altitudine;
    float velocita;
}
```

```
struct Aereo aereo;
```

Relazioni tra le entità (tra le strutture dati) - In questa sezione devono essere descritte le relazioni tra le entità coinvolte nell'applicazione, facendo anche riferimento alle strutture dati usate per rappresentarle. Possono essere utilizzati dei grafici che collegano le entità rappresentate da dei rettangoli. I collegamenti che finora sono stati usati sono:

- **Aggregazione**  Questa relazione prevede che un'entità abbia (HAS-A) un insieme di altre entità fungendo da contenitore per la collezione. I componenti contenuti possono esistere senza il contenitore e possono essere associati a diversi contenitori allo stesso tempo. Ad esempio, un cliente HA un'aggregazione di metodi di pagamento, ma l'eventuale eliminazione del cliente non provoca anche l'eliminazione del metodo di pagamento.
- **Composizione**  Una relazione di composizione si ha quando un'entità è composta da una o più istanze di un'altra entità. La differenza tra questa relazione e l'aggregazione è che il componente può esistere solo come parte del contenitore. Anche questa relazione è comunque una relazione di tipo HAS-A. Ad esempio, il sistema informativo di un rivenditore online di prodotti HA delle collezioni di prodotti, clienti e metodi di pagamento che sono delle sue parti e, se viene meno il sistema informativo aziendale le varie collezioni in esso contenute cessano di esistere con esso.
- **Generalizzazione/Specializzazione**  Una relazione di specializzazione si ha quando un'entità che rappresenta un concetto generico viene specializzata con l'aggiunta di ulteriori attributi, al fine di rappresentare una nuova entità che contiene sia le caratteristiche della generalizzazione, sia le nuove caratteristiche della

specializzazione. Ad esempio, l'entità generica che rappresenta una persona può essere usata per rappresentare una parte delle caratteristiche di un insegnante. L'entità dell'insegnante ingloba tutte le caratteristiche di una persona perché È UNA (IS-A) persona, ma aggiunge anche ulteriori caratteristiche specifiche dell'insegnante, come ad esempio il numero di matricola o le abilitazioni all'insegnamento delle varie discipline.

Altre tipologie di relazioni che saranno introdotte quando sarà necessario sono:

- Dipendenza>
- Associazione —>
- Generalizzazione —▷

Casi d'uso

Caso d'uso principale - In questa sezione si descrive come viene usata l'applicazione dal punto di vista di un utente che ha fatto una richiesta al sistema. Un caso d'uso cattura la sequenza di eventi *visibili* attraverso cui il sistema deve procedere per rispondere ad una singola richiesta dell'utente. I casi d'uso sono derivati dai colloqui effettuati con gli [stakeholder](#) e, per loro natura sono destinati a cambiare frequentemente, quindi è bene mantenerli semplici. Ogni caso d'uso presenta almeno un *caso principale* che descrive la risposta del sistema quando tutto procede correttamente. Per ogni *Caso d'uso principale* è possibile prevedere anche degli esempi di ciò che deve essere visualizzato e inserito.

Casi d'uso alternativi - Spesso nei *Casi d'uso principali* sono presenti dei *Casi d'uso alternativi*, che si verificano quando il *Caso d'uso principale* può presentare dei problemi da gestire. Nei *Casi d'uso alternativi* è necessario indicare quando si verificano e cosa bisogna fare per recuperare la situazione anomala. Anche per i *Casi d'uso alternativi* si prevedono degli esempi di ciò che viene visualizzato dal programma.

Principi di programmazione usati - In questa sezione devono essere indicati e spiegati i principi di programmazione usati, come ad esempio

- la [programmazione modulare](#) per separare l'applicazione in moduli distinti;
- il [principio della separazione delle competenze](#) dei moduli (*SoC - Separation of Concern*);
- il [principio della singola responsabilità](#) delle funzioni (*SRP - Single Responsibility Principle*);
- l'attuazione del [principio di incapsulamento](#) delle (*Encapsulation principle*) delle strutture dati e delle funzioni nei moduli di competenza;
- il [principio Open-Close](#) (*Open-Closed principle*) sul concetto di estensione del codice di un modulo invece della sua modifica.

Codice dell'applicazione - In questa sezione viene riportato il codice dell'applicazione, se richiesto.

145. Partecipi alla progettazione di un gioco di automobili e ti devi occupare del parco macchine disponibile per il gioco. Di ogni auto si vogliono memorizzare la marca, il modello, il colore, la cilindrata e la velocità. Crea un'applicazione che permetta di

definire il parco macchine, caricare i dati delle macchine e visualizzare i dati come elenco.

146. Partecipi alla progettazione di un simulatore di volo e ti devi occupare di definire gli aeroplani disponibili nel simulatore. Di ogni aereo si vogliono memorizzare la marca, il modello, il numero dei motori, l'altitudine e la velocità. Crea un'applicazione che permetta di definire l'insieme degli aerei, caricare i dati degli aerei e visualizzare i dati come elenco ordinato per marca e modello. Inoltre l'applicazione deve permettere di selezionare una marca e visualizzare tutti i modelli di quella marca. Gestire le operazioni tramite un menù.
147. La tua azienda ha ricevuto l'appalto per la creazione di una nuova applicazione di prenotazione dei voli di aerei. Tu sei stato incaricato inizialmente di svolgere la progettazione descrivendo tutti i punti indicati nella [documentazione spiegata in classe](#), seguendo le indicazioni fornite successivamente che derivano dall'intervista fatta al committente.

L'applicazione deve modellare due entità distinte, gli aerei e i passeggeri.

Di ogni aereo si vogliono memorizzare il codice dell'aereo (alfanumerico) e l'elenco dei posti con l'indicazione se il posto è occupato o libero. *Per semplificare l'esercizio vengono tralasciati i luoghi di partenza e di arrivo, l'ora di partenza e di arrivo, e molte altre caratteristiche tipiche della prenotazione dei voli.*

Di ogni passeggero si vogliono memorizzare il nome, il cognome, l'indirizzo email, il numero di telefono e la data di nascita (tutti campi alfanumerici).

Ogni passeggero deve essere collegato al relativo posto e spetta a te trovare una modalità per creare questo collegamento in modo tale che risulti poco dispendiosa in termini di spazio occupato e tempo di accesso.

L'applicazione deve permettere ad un cliente di prenotare un posto nell'aereo, nel caso sia ancora possibile, usando un menù di prenotazione. Se risultano disponibili ancora dei posti, l'applicazione chiede tutti i dati al cliente per memorizzarli e lo associa al primo posto libero trovato, comunicandolo al cliente. Se non sono disponibili altri posti liberi lo comunica al cliente.

L'applicazione dovrebbe gestire più aerei, ma inizia a creare la progettazione per gestire un singolo aereo con più passeggeri prenotati. Una volta terminata questa fase, amplia la progettazione per gestire più aerei.

Per gestire più aerei deve essere implementato un menù superiore che presenta l'elenco dei codici degli aerei in ordine alfabetico, permettendo al cliente di scegliere quello desiderato. Una volta scelto si procede con il menù di prenotazione descritto precedentemente.

148. Terminata la gestione delle prenotazioni degli aerei, l'aeroporto commissiona alla ditta per cui lavori anche lo sviluppo dell'applicazione di gestione dei voli per caricare i dati degli aerei e gestire le prenotazioni fatte dai clienti. Vista la tua bravura, la ditta ti commissiona lo sviluppo anche di questa progettazione descrivendo tutti i punti indicati nella [documentazione spiegata in classe](#), seguendo le indicazioni fornite successivamente che derivano dall'intervista fatta al committente.

In questo caso l'utilizzatore è un addetto dell'aeroporto che si occupa di caricare tutti i dati degli aerei: codice dell'aereo e numero di posti per ogni aereo. L'applicazione deve automaticamente inizializzare tutti i posti dell'aereo come liberi.

L'addetto deve anche poter richiedere la visualizzazione del numero dei posti liberi e occupati di un aereo e l'elenco dei passeggeri di un aereo con il relativo posto che occupano.

149. Alla ditta per cui lavori viene richiesta una variazione dell'applicazione di prenotazione degli aerei su cui stai lavorando; è una specifica dei requisiti a cui hanno pensato solo ora. Dato che ti stai occupando tu delle progettazioni di prenotazioni e di gestione dei voli da parte del personale, la ditta ti assegna l'incarico di modificare entrambe le progettazioni in modo da includere quanto emerso dall'intervista. Riprendi le due progettazioni e modificali opportunamente.

Ogni cliente che intende prenotare un posto in un aereo deve poter scegliere il posto sull'aereo stesso, tra quelli rimasti liberi.

L'applicazione deve mostrare al cliente l'elenco dei soli posti liberi e il cliente inserisce il numero del posto che vuole prenotare.

150. Implementa l'esercizio di prenotazione dei posti negli aerei.

151. Si vogliono organizzare i dati relativi ai risultati di una corsa campestre sulla distanza dei 1000 metri. Di ogni atleta si intende memorizzare il numero di gara, il nominativo e il tempo in secondi impiegato per portare a termine la prova. I dati dovranno essere memorizzati in un singolo array definendo un nuovo tipo per gestire i diversi dati. Si crei un'applicazione in grado di caricare gli atleti e visualizzarli.

152. Creare la progettazione di un'applicazione di gestione di una rubrica telefonica. La rubrica colleziona al suo interno tutti i dati dei contatti.

Ogni contatto deve essere identificato in modo univoco all'interno della rubrica tramite un codice numerico che viene incrementato automaticamente dall'applicazione quando viene inserito un nuovo contatto.

Di ogni contatto si vogliono mantenere i seguenti dati: nome, cognome, società, numero di telefono, indirizzo email, indirizzo residenza.

La rubrica deve permettere di aggiungere un nuovo contatto, rimuoverlo, visualizzare l'elenco dei contatti in ordine alfabetico di cognome e di nome e, di visualizzare tutti i dati di un contatto fornito il nome e il cognome da cercare.

Quando un contatto viene rimosso dalla rubrica telefonica il codice numerico che era associato ad esso non deve essere più riutilizzato.

Sviluppa la progettazione descrivendo tutti i punti indicati nella [documentazione spiegata in classe](#). Osserva che in questo problema può essere plausibile prevedere una struct che modella un contatto con all'interno una struct che modella la persona con i suoi dati. Decidi in autonomia se seguire questa strada o basarti su qualche cosa di simile alla gestione delle prenotazioni dei voli.

153. Sei stato consultato per la creazione di un'applicazione del Fantacalcio e devi progettare le strutture dati per mantenere i dati delle squadre e degli allenatori. Una volta definiti i nuovi tipi, per testarli dovrai creare una semplice applicazione che permetta di caricare i dati e visualizzarli.

Una squadra di calcio è identificata dalle seguenti informazioni:

- a. nome della squadra;
- b. colore della maglia;
- c. punteggio corrente;
- d. allenatore.

L'allenatore è una persona caratterizzata da:

- a. nome e cognome;
- b. numero di titoli vinti.

154. Una **banca** vuole gestire i suoi **clienti** e i **conti correnti**, mantenendo traccia delle **transazioni** effettuate in essi e quindi ti commissiona un software atto allo scopo.

Per gestire l'applicazione decidi di separare in moduli diversi le competenze di gestione di ogni entità concettuale coinvolta dalla banca, usando anche librerie che hai già sviluppato. In questo modo sei in grado di gestire problemi più semplici che, uniti insieme, costituiranno la tua applicazione finale.

Per la gestione delle **transazioni** decidi di sviluppare una nuova struct perché non hai mai dovuto gestire un'entità di questo tipo. Procedi quindi con la creazione della struct dentro una nuova libreria, considerando che per ogni transazione si vogliono memorizzare:

- a. il numero della transazione che per ogni conto corrente deve essere un numero incrementale che parte sempre da 1;
- b. l'importo della transazione che deve essere un valore positivo per i depositi e un valore negativo per i prelievi;
- c. la data della transazione;
- d. la descrizione della transazione.

Le possibili descrizioni di una transazione sono fissate e coprono le normali attività bancarie di un correntista, come ad esempio *Apertura conto*, *Chiusura conto*, *Carta di credito*, *Bonifico*, *Pagamento POS*, *Versamento contanti presso ATM*, *Pagamento utenze*, *Accredito stipendio*, e così via.

Per la gestione dei **clienti** decidi di usare una struct che hai già creato per altri lavori, la struct *Persona* utilizzata nel programma del Fantacalcio, in quanto i dati relativi ai clienti della banca corrispondono a quelli usati per gli allenatori del Fantacalcio. *Hint - Se non è stata ancora creata la libreria persona.h la si crei, dovrà contenere i dati anagrafici principali di una persona.*

La gestione dei **conti correnti** è invece nuova e quindi decidi di creare una nuova struct all'interno di una nuova libreria. Per ogni conto corrente dovranno essere considerati i seguenti dati:

- a. il codice IBAN;
- b. l'elenco degli intestatari, tutti clienti della banca;

c. la lista delle transazioni effettuate nel conto.

Nell'applicazione si suppone che un cliente possa essere intestatario di più conti correnti e, un conto corrente deve avere almeno un intestatario, ma può averne anche più di uno.

Per testare il funzionamento delle nuove struct scrivi un programma che permetta di gestire un numero arbitrario di conti correnti, con la relativa lista di intestatari e transazioni, prevedendo il caricamento di tutti i dati e la successiva visualizzazione.

Infine prova a visualizzare l'elenco delle transazioni di un conto e il relativo saldo.

La memoria dinamica

La gestione dinamica della memoria permette di strutturare applicazioni dinamiche e flessibili. Leggi i seguenti articoli e guarda i video per comprendere le diverse sfaccettature della sua gestione.

Array numerici dinamici

C Tutorial

- 19.C Programming Arrays by [Programiz](#)
 - a. [C Pointers](#)
 - b. [Relationship Between Arrays and Pointers](#)
 - c. [C Pass Addresses and Pointers](#)
 - d. [C Dynamic Memory Allocation](#)
- 20.[Dynamic Array in C](#) by Prateek Narang, [Scaler Topics](#)
- 21.[Dynamic Array in C](#), [GeegsForGeeks](#)
- 22.VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)
 - a. [Pointer Notation](#)
 - b. [array vs &array Pointers Difference Explained](#)
 - c. [Introduction to Pointers](#)
 - d. [How To Return An Array From A Function](#)
 - e. [malloc vs calloc Differences Explained](#)
 - f. [realloc Basics](#)
 - g. [Dynamic Memory Allocation](#)

Esempio di gestione di un array dinamico di numeri

L'applicazione si occupa di gestire l'input di un numero arbitrario di valori float, li ordina in ordine crescente e li visualizza.

Per gestire l'applicazione è stata creata una libreria per la gestione degli array, `arraymanage.c`, che contiene al suo interno tutte le funzioni utili per la gestione di un array di float. Per rendere modulare la libreria è stato creato anche un header, `arraymanage.h`, che contiene l'interfaccia pubblica della libreria e che viene usato dall'applicazione come unico punto di collegamento con la definizione effettiva delle funzioni di gestione degli array.

L'**allocazione dell'array** avviene **dinamicamente** usando la funzione `malloc()` e viene **creato nella memoria Heap**. Una volta **terminato il suo utilizzo** lo spazio di **memoria** dell'array viene **deallocato** usando la funzione `free()`.

L'array viene dichiarato nel `main()` come semplice puntatore a float, senza necessità di indicare una dimensione. Questo è possibile perché in effetti qualsiasi array punta sempre all'indirizzo di memoria della prima cella di memoria dell'array e, in questo caso non si fa eccezione alla regola, il puntatore fa sempre riferimento alla prima cella dell'array.

Il puntatore array è una variabile locale al `main()` e, come tale, viene memorizzato nella memoria Stack del `main()`. L'indirizzo di memoria contenuto nella variabile array è invece un indirizzo della memoria Heap e corrisponde alla prima cella di memoria dell'array di float.

L'array deve essere caricato nella memoria Heap proprio per la sua caratteristica dinamica: potendo essere ridimensionato a piacimento la memoria Heap è l'unica che permette questa operazione. Si lascia come esercizio l'estensione della libreria `arraymanage` per includere una funzione di ridimensionamento dell'array.

Nelle funzioni che prevedono come **parametro formale l'array, il passaggio del parametro attuale avviene per valore**, cioè l'indirizzo memorizzato nel puntatore del `main()` viene copiato nel parametro formale della funzione. Questo avviene per le seguenti funzioni:

- Prototipo funzione: `void caricaValori(float *array, int n);`
Invocazione funzione: `caricaValori(array, n);`
- Prototipo funzione: `void ordinaValori(float *array, int n);`
Invocazione funzione: `ordinaValori(array, n);`
- Prototipo funzione: `void deallocaArrayFloat(float *array);`
Invocazione funzione: `deallocaArrayFloat(array);`

La funzione `stampaValori()` ha il medesimo comportamento delle altre funzioni, ma il parametro formale viene dichiarato come `const float*`. Il modificatore `const` indica che il contenuto dell'array passato come parametro è immutabile. Il parametro attuale array viene comunque passato per valore:

- Prototipo funzione: `void stampaValori(const float* array, int n)`
Invocazione funzione: `stampaValori(array, n);`

In tutte queste funzioni il parametro formale `float* array` prevede che venga passato un indirizzo di memoria, quindi corrisponde formalmente ad un passaggio di parametro per indirizzo, ma nel `main()` viene passato il contenuto di un puntatore, non l'indirizzo di memoria del puntatore. Quindi ciò che viene copiato nel parametro formale è consistente con un passaggio di parametro per indirizzo, ma di fatto viene copiato il contenuto di un puntatore, risultando in un passaggio di parametro per valore.

main.c

```
#include "lib/arraymanage/arraymanage.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
    int n;
    float *array;

    // Richiedi all'utente la dimensione dell'array
    printf("Inserisci la dimensione dell'array: ");
    scanf("%d", &n);
    // Alloca dinamicamente l'array di float
    array = allocaArrayFloat(n);
    // Carica i valori nell'array
    caricaValori(array, n);
    // Stampa i valori inseriti nell'array
    stampaValori(array, n);
    // Ordina l'array in ordine crescente
    ordinaValori(array, n);
    // Stampa i valori inseriti nell'array
    stampaValori(array, n);
    // Dealloca l'array dinamico
    deallocaArrayFloat(array);

    return 0;
}
```

arraymanage.h

```
float* allocaArrayFloat(int n);
void caricaValori(float *array, int n);
void stampaValori(const float *array, int n);
void ordinaValori(float *array, int n);
void deallocaArrayFloat(float *array);
```

arraymanage.c

```
#ifndef INCLUDE_H
#define INCLUDE_H
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#endif // INCLUDE_H

/**
 * Alloca nello heap uno spazio di memoria pari a n volte la dimensione di un float
 * @param n numero di celle da allocare
 * @return il puntatore all'array allocato
 */
```

```
*/
float *allocaArrayFloat(int n) {
    // Alloca dinamicamente un array di float di dimensione n
    float *array = malloc(n * sizeof(float));

    // Verifica se l'allocazione è avvenuta correttamente
    if (array == NULL) {
        printf("Errore: memoria non disponibile.");
        exit(1); // Esci con codice di errore
    }
    // Se l'allocazione è andata a buon fine restituisce il valore del puntatore
    return array;
}

/**
 * Carica i valori nell'array
 * @param *array puntatore all'array da caricare
 * @param n dimensione dell'array
 */
void caricaValori(float *array, int n) {
    // Inserisci i valori nell'array
    printf("Inserisci %d valori nell'array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%f", &array[i]);
    }
}

/**
 * Visualizza il contenuto dell'array
 * @param *array puntatore all'array da visualizzare
 * @param n dimensione dell'array
 */
void stampaValori(const float *array, int n) {
    // Stampa i valori inseriti nell'array
    printf("Valori nell'array:\n");
    for (int i = 0; i < n; i++) {
        printf("%.2f ", array[i]);
    }
    printf("\n");
}

/**
 * Ordinamento crescente dell'array usando l'algoritmo bubble sort
 * @param *array puntatore all'array da ordinare
 * @param n dimensione dell'array
 */
void ordinaValori(float *array, int n) {
    bool swapped;
    for (int i = 0; i < n; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                float temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```

```
        swapped = true;
    }
}
// Se non avvengono scambi, interrompi l'algoritmo
if (swapped == false) break;
}
}

/**
 * Deallocazione dell'array
 * @param *array puntatore all'array da deallocare
 */
void deallocaArrayFloat(float *array) {
    // Libera la memoria allocata per l'array
    free(array);
}
```

Una possibile esecuzione dell'applicazione genera i seguenti output a fronte degli input forniti.

Inserisci la dimensione dell'array: 5

Inserisci 5 valori nell'array:

7

2

4

9

8

Valori nell'array:

7.00 2.00 4.00 9.00 8.00

Valori nell'array:

2.00 4.00 7.00 8.00 9.00

Esercizi

155. Riprendi l'esercizio relativo al PM10 delle città e crea un array dinamico per gestire solo il PM10. L'applicazione deve permettere di caricare il valore del PM10 per un numero di città scelto dall'utente, deve visualizzare il PM10 massimo, il PM10 minimo, la media di tutti i PM10 e l'elenco dei PM10 in ordine decrescente.
- Usa un menù per permettere la scelta delle diverse opzioni considerando che bisogna prima caricare i valori del PM10 prima di poter scegliere le altre opzioni.
- Usa la memoria dinamica e tutti i principi della programmazione modulare visti finora.

Stringhe dinamiche e array di stringhe

C Tutorial

23. VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#) and [C Programming Example](#)
- a. [How To Return A String From A Function](#)
 - b. [Dynamically Allocate Memory For An Array Of Strings](#)
 - c. [NULL Pointer | C Programming Tutorial](#)
24. [C Strings - Declaring Strings in C](#) by Prateek Narang, [Scaler Topics](#)
25. [Short answer](#) to [Arbitrary length string in C](#) by [Palec](#), [stackoverflow](#)
26. [The top rated answer](#) to [const char * const versus const char *?](#) by [jazzpi](#), [stackoverflow](#)
- char* the_string** : I can change which char the_string points to, and I can modify the char to which it points.
- const char* the_string** : I can change which char the_string points to, but I cannot modify the char to which it points.
- char* const the_string** : I cannot change which char the_string points to, but I can modify the char to which it points.
- const char* const the_string** : I cannot change which char the_string points to, nor can I modify the char to which it points.
27. [The top rated answer](#) to [Char ** usage and printing](#) by [Simon](#), [stackoverflow](#)
28. [The top rated answer](#) to [Dynamically create an array of strings with malloc](#) by [MByD](#), [stackoverflow](#)

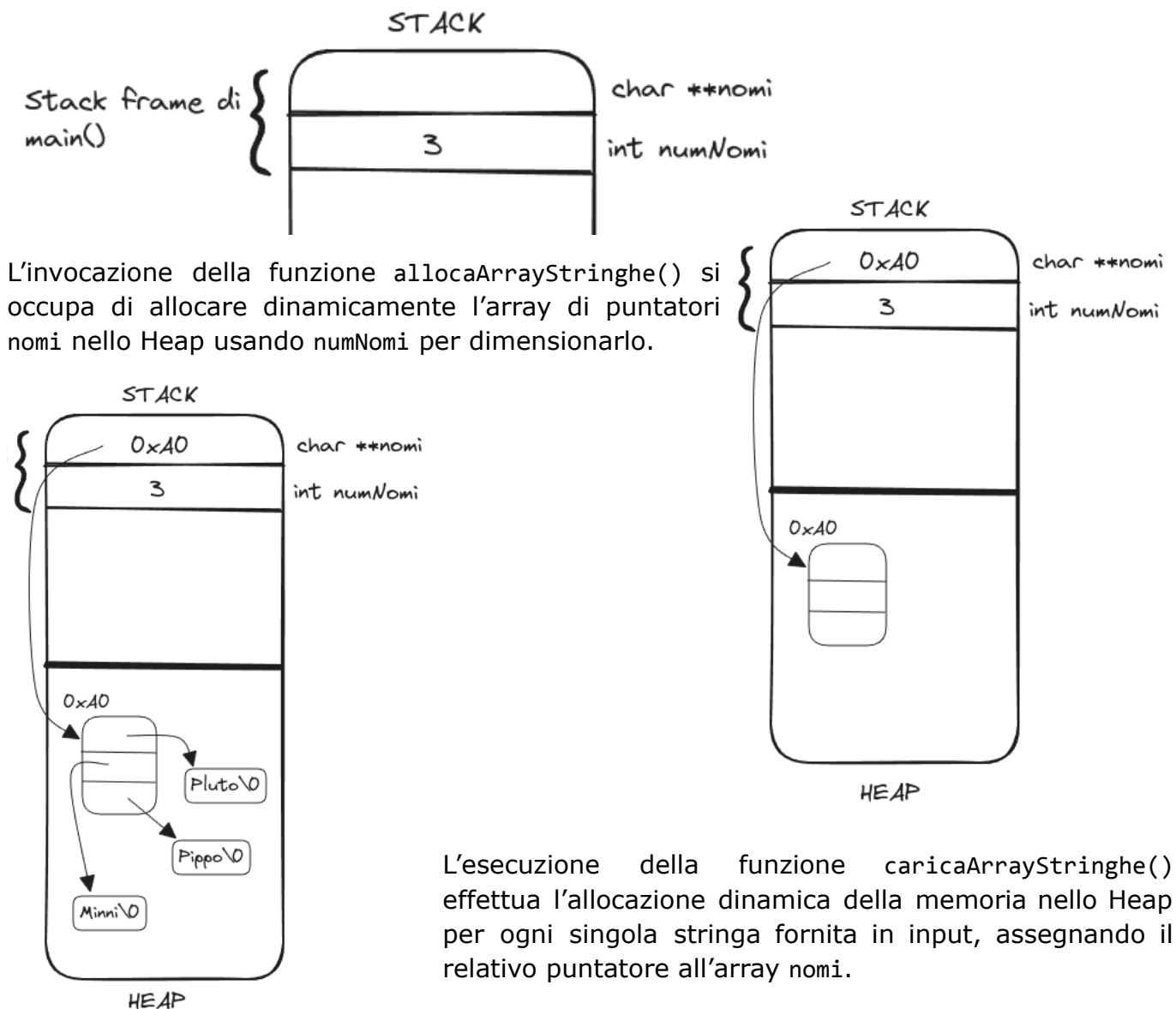
Esempio di array dinamico di stringhe

La seguente applicazione permette di allocare dinamicamente un array di stringhe, ordinandole lessicograficamente e visualizzandole.

Nella funzione `main()` vengono usate due variabili locali definite come `char **nomi` atta a contenere l'elenco dei nomi da allocare e, `int numNomi` usata per indicare il numero di nomi da caricare.

La variabile `char **nomi` è un puntatore di puntatori in quanto risulta essere un array di stringhe. Dato che punta ad un array nella memoria Heap deve essere un puntatore e, ogni sua singola posizione è costituita da puntatori a stringhe dinamiche sempre allocate nella memoria Heap. Da ciò deriva la doppia referenziazione, cioè `nomi` è un array di puntatori.

Con l'avvio di `main()` viene allocato nella memoria Stack lo stack frame della funzione che prevede l'allocazione delle due variabili locali¹⁰.



L'esecuzione della funzione `caricaArrayStringhe()` effettua l'allocazione dinamica della memoria nello Heap per ogni singola stringa fornita in input, assegnando il relativo puntatore all'array `nomi`.

La funzione `visualizzaArrayStringhe()` effettua la visualizzazione delle stringhe puntate da `nomi`, mentre `deallocaArrayStringhe()` libera la

¹⁰ Le immagini sono state create usando [Excalidraw](#).

memoria Heap prima deallocando ogni singola stringa puntata da nomi e, successivamente deallocando nomi stesso.

main.c

```
#include <stdio.h>
#include "lib/stringhe/stringhe.h"

/**
 * L'applicazione permette di caricare un array dinamico di stringhe e di
 * visualizzarle.
 */
int main(void) {
    char **nomi;
    int numNomi;

    printf("Inserisci il numero di nomi: ");
    scanf("%d", &numNomi);
    while (getchar() != '\n')
        ;
    nomi = allocaArrayStringhe(numNomi);
    caricaArrayStringhe(nomi, numNomi, "Caricare i nomi\n");
    visualizzaArrayStringhe(nomi, numNomi, "Elenco nomi non ordinato\n");
    ordinaArrayStringhe(nomi, numNomi);
    visualizzaArrayStringhe(nomi, numNomi, "Elenco nomi ordinato\n");
    deallocaArrayStringhe(nomi, numNomi);

    return 0;
}
```

stringhe.h

```
#pragma once
char **allocaArrayStringhe(int numStringhe);
void caricaArrayStringhe(char **array, int numStringhe, const char *messaggio);
void deallocaArrayStringhe(char **array, int numStringhe);
void visualizzaArrayStringhe(char **array, int numStringhe, const char *messaggio);
```

stringhe.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define DIM 4096

/**
 * Alloca dinamicamente un array di stringhe.
 * @param numStringhe Numero di stringhe da allocare
 * @return l'indirizzo di memoria dell'array di stringhe allocato nello Heap
 */
char **allocaArrayStringhe(int numStringhe) {
    char **array;
    array = malloc(numStringhe * sizeof(char *));
    if (array == NULL) {
        printf("Errore nell'allocazione dell'array\n");
        exit(1);
    }
}
```

```
    return array;
}

/**
 * Alloca dinamicamente una stringa.
 * Questa funzione non viene esposta nell'interfaccia stringe.h perché è una
 * funzione di servizio e non deve essere usata al di fuori di questo file.
 * @param lunghezza - Lunghezza della stringa da allocare
 * @return l'indirizzo di memoria della stringa
 */
char *allocaStringa(int lunghezza) {
    char *stringa;
    stringa = malloc(lunghezza * sizeof(char));
    if (stringa == NULL) {
        printf("Errore nell'allocazione della stringa\n");
        exit(1);
    }
    return stringa;
}

/**
 * Carica l'array di stringhe dinamiche.
 * @param array - Array di stringhe dinamiche
 * @param numStringhe - Numero di stringhe da caricare nell'array
 * @param messaggio - Messaggio da visualizzare per indicare cosa caricare
 */
void caricaArrayStringhe(char **array, int numStringhe, const char *messaggio) {
    char temp[DIM];
    printf("%s", messaggio);
    for (int i = 0; i < numStringhe; i++) {
        printf("%d°: ", i + 1);
        fgets(temp, DIM, stdin);
        int length = strlen(temp);
        temp[length - 1] = '\0';
        // La funzione allocaStringa() restituisce solo l'indirizzo di memoria nello
        // Heap dello spazio allocato, di dimensione pari a length, ma la stringa
        // presente in temp non è ancora stata copiata in questo spazio di memoria
        array[i] = allocaStringa(length);
        // In questo momento viene copiata la stringa salvata in temp, memorizzata nello
        // Stack, all'interno dello spazio di memoria puntato da array[i] nello Heap
        // strcpy(array[i], temp);
        strncpy(array[i], temp, length);
    }
}

/**
 * Ordina l'array di stringhe seguendo un ordinamento lessicografico crescente.
 * @param array - Array da ordinare
 * @param numStringhe - Numero di stringhe presenti nell'array
 */
void ordinaArrayStringhe(char **array, int numStringhe) {
    for(int i=0; i<numStringhe; i++){
        bool swapped;
        for(int j=0; j<numStringhe-i-1; j++){
            if(strcmp(array[j], array[j+1]) > 0){
```

```
        char *temp = array[j];
        array[j] = array[j+1];
        array[j+1] = temp;
        swapped = true;
    }
}
if (swapped == false) break;
}
}

/**
 * Dealloca l'array di stringhe deallocando prima ogni singola stringa e poi
 * l'intero array.
 * @param array - Array da deallocare
 * @param numStringhe - Numero di stringhe presenti nell'array
 */
void deallocaArrayStringhe(char **array, int numStringhe) {
    for (int i = 0; i < numStringhe; i++) {
        free(array[i]);
    }
    free(array);
}

/**
 * Visualizza le stringhe puntate dall'array di stringhe.
 * @param array - Array di stringhe dinamiche
 * @param numStringhe - Numero di stringhe da caricare nell'array
 * @param messaggio - Messaggio da visualizzare per indicare cosa si sta
 *                   visualizzazando
 */
void visualizzaArrayStringhe(char **array, int numStringhe,
                             const char *messaggio) {
    printf("%s", messaggio);
    for (int i = 0; i < numStringhe; i++) {
        printf("%s\n", array[i]);
    }
}
```

Esercizi

156. Riprendi l'esercizio relativo al PM10 delle città gestito con la memoria dinamica e, crea un ulteriore array dinamico per gestire solo le città. L'applicazione deve permettere di caricare i nomi delle città parallelamente al corrispondente valore del PM10, per un numero di città scelto dall'utente. Inoltre l'applicazione deve visualizzare la città con il PM10 massimo, la città con il PM10 minimo, la media di tutti i PM10 e l'elenco delle città con i corrispondenti PM10 in ordine decrescente.

Usa un menù per permettere la scelta delle diverse opzioni considerando che bisogna prima caricare i valori delle città e dei corrispondenti PM10 prima di poter scegliere le altre opzioni. Usa la memoria dinamica e tutti i principi della programmazione modulare visti finora.

Se la gestione parallela dei due array dinamici ti risulta ancora troppo difficile, prova prima a gestire solo l'array delle città, caricando i nomi e visualizzando il loro contenuto in ordine crescente di nome città. Successivamente unisci i due array e gestisci l'applicazione completa.

Allocazione dinamica di struct

C Tutorial

29. [C structs and Pointers](#) by [Programiz](#)

Esempio di allocazione dinamica di una struct

Le strutture costituiscono uno strumento per aggregare caratteristiche che definiscono una medesima entità.

Nell'esempio seguente si vuole astrarre il concetto di una persona e viene creata la struttura **Persona** che definisce un **nuovo tipo** da utilizzare nel programma.

```
struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
};
```

Una volta definita la struttura che astrae il concetto di una persona, è possibile sviluppare un'applicazione che permette di allocare un oggetto di tipo Persona.

L'allocazione dinamica di un oggetto di tipo Persona prevede la dichiarazione della variabile puntatore ***persona** e l'uso della funzione `malloc()` che deve dimensionare l'area di memoria nello Heap usando il nuovo tipo Persona.

```
struct Persona *persona = malloc(sizeof(struct Persona));
```

Dato che la variabile **persona** è un puntatore, per identificare i campi della struttura è possibile utilizzare l'operatore **->** (*arrow*) invece dell'operatore **.** (*dot*). L'uso dell'operatore **->** (*arrow*) semplifica la gestione di una variabile puntatore a **struct**, funzionando in modo analogo all'operatore **.** (*dot*). Il seguente esempio mostra solo l'uso dell'operatore **->** (*arrow*).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LEN_NOMINATIVO 30
#define LEN_INDIRIZZO 50

struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
};

int main(void) {
    // Allocazione di un oggetto di tipo struct Persona
    struct Persona *persona = malloc(sizeof(struct Persona));
    if (persona == NULL) {
        printf("Errore nell'allocazione della persona\n");
        exit(1);
    }

    // Caricamento dei dati di una persona
    printf("Id: ");
    scanf("%d", &persona->id);
    getchar();
```

```
printf("Nominativo: ");
fgets(persona->nominativo, LEN_NOMINATIVO, stdin);
persona->nominativo[strlen(persona->nominativo) - 1] = '\0';
printf("Indirizzo: ");
fgets(persona->indirizzo, LEN_INDIRIZZO, stdin);
persona->indirizzo[strlen(persona->indirizzo) - 1] = '\0';

// Visualizzazione dei dati della persona
printf("%d - %s, %s\n", persona->id, persona->nominativo,
        persona->indirizzo);

free(persona);

return 0;
}
```

Una esecuzione dell'applicazione produce il seguente risultato.

```
Id: 1
Nominativo: Minni
Indirizzo: Via Roma, 5
1 - Minni, Via Roma, 5
```

Struct e programmazione modulare

Strutture nascoste

L'utilizzo della programmazione modulare permette di gestire un'applicazione in modo tale che le modifiche di un modulo non vadano ad influenzare il resto del programma. È quindi importante utilizzare questa tecnica di programmazione anche con le struct dinamiche.

L'uso di un modulo separato nel quale porre la definizione di una struct permette di introdurre concretamente il concetto di **strutture nascoste** o, più in generale di **information hiding**. Quando una struttura viene definita in un modulo separato, per poterla utilizzare all'interno della funzione `main()` è necessario includere un file header (`.h`) che contenga la sola dichiarazione della struct, senza la presenza dei suoi campi, dichiarando di fatto un tipo incompleto.

Nascondere in questo modo una struttura può sembrare un'operazione strana, ma in realtà non espone tutte le caratteristiche di una struttura e una pratica che permette di evitare errori di programmazione e di rendere debolmente accoppiati i moduli di un programma.

Non avendo a disposizione la definizione completa della struct, la funzione `main()` non è in grado di sapere quali sono i campi che la compongono, quindi è necessario definire nel modulo anche le funzioni che permettono di accedere ai singoli campi della struttura, riportando nell'header (`.h`) le loro dichiarazioni.

Questa modalità di programmare permette di aggiungere il principio di **incapsulamento** ([Encapsulation principle](#)) alla programmazione modulare vista finora.

Il modulo che gestisce la struct conterrà la sua definizione e la definizione di tutte le funzioni necessarie per accedere ai campi. Il file header (`.h`) da includere nella funzione `main()` conterrà invece le sole dichiarazioni, sia della struct, sia delle funzioni.

L'**incapsulamento** è un principio molto potente nella programmazione perché permette di definire una precisa **interfaccia** con la quale l'applicazione può interagire con il modulo incluso. Il file header (`.h`) **espone esclusivamente** le funzioni che si desidera vengano utilizzate, quindi è possibile permettere l'accesso solo ad alcuni campi della struct, nascondendo completamente altri che possono risultare utili solo all'interno del modulo. Inoltre si può decidere se fornire o meno la possibilità di modificare alcuni campi di un oggetto definito con il tipo della struct, prevedendo o meno le funzioni che permettono questa azione.

Dovrebbe essere chiaro che l'uso di un file header (`.h`) che costituisce un'**interfaccia di comunicazione**, permette di applicare completamente i concetti di basso accoppiamento ([Coupling](#)) e alta coesione ([Cohesion](#)) tra i moduli dell'applicazione, permettendo anche di perseguire il principio di separazione delle competenze dei moduli ([Separation of concern principle](#) - *SoC principle*)¹¹.

¹¹ Leggere nuovamente l'intera sezione [Dalla programmazione strutturata alla programmazione modulare](#) alla luce di tutte le nuove competenze acquisite finora.

C Tutorial

30. [Encapsulation \(computer programming\)](#) by [Wikipedia](#). Vedere solo l'esempio in C.

Esempio di programmazione modulare con strutture nascoste

In questo paragrafo viene ripreso l'esempio precedente, modificandolo in modo da utilizzare la programmazione modulare ed applicare sia il concetto di **information hiding**, sia quello di **incapsulamento**.

Viene quindi previsto un nuovo modulo, `persona.c`, che contiene la definizione della struttura `struct Persona`, la funzione per creare un nuovo oggetto di tipo `Persona`

```
struct Persona *newPersona(int id, char *nominativo, char *indirizzo)
```

e le funzioni per accedere al contenuto dei singoli campi dell'oggetto

```
int getId(struct Persona *persona)
char *getNominativo(struct Persona *persona)
char *getIndirizzo(struct Persona *persona)
```

Il modulo prevede anche una sola funzione per modificare l'indirizzo

```
void setIndirizzo(struct Persona *persona, const char *indirizzo)
```

ma non prevede la modifica dell'id e del nominativo. In questo modo il modulo controlla completamente l'accesso ad un qualsiasi oggetto di tipo `Persona`.

Per permettere alla funzione `main()` di gestire un oggetto di tipo `struct Persona` è necessario creare il file `persona.h` nel quale vengono poste le dichiarazioni della struttura e delle funzioni.

main.c

```
#include <stdio.h>
#include <string.h>
#include "lib/persona/persona.h"

#define LEN_NOMINATIVO 30
#define LEN_INDIRIZZO 50

int main(void) {
    struct Persona *persona;
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];

    // Caricamento dati per creare una nuova persona
    printf("Id: ");
    scanf("%d", &id);
    getchar();
    printf("Nominativo: ");
    fgets(nominativo, LEN_NOMINATIVO, stdin);
    nominativo[strlen(nominativo) - 1] = '\0';
    printf("Indirizzo: ");
    fgets(indirizzo, LEN_INDIRIZZO, stdin);
    indirizzo[strlen(indirizzo) - 1] = '\0';

    // Creazione di una nuova persona
    persona = newPersona(id, nominativo, indirizzo);
```

```
// Visualizzazione dei dati della persona
printf("%d - %s, %s\n", getId(persona), getNominativo(persona),
      getIndirizzo(persona));

// Modifica dell'indirizzo della persona
bool flag = setIndirizzo(&persona, "Via Roma, 1");
if (!flag) printf("Indirizzo non modificato\n");

// Visualizzazione dei dati della persona modificati
printf("%d - %s, %s\n", getId(persona), getNominativo(persona),
      getIndirizzo(persona));

// Deallocazione di persona
freePersona(persona);

return 0;
}
```

persona.h

```
#pragma once
```

```
struct Persona; // Struttura nascosta, detto anche tipo incompleto
```

```
struct Persona *newPersona(int id, char *nominativo, char *indirizzo);
int getId(struct Persona *persona);
char *getNominativo(struct Persona *persona);
char *getIndirizzo(struct Persona *persona);
bool setIndirizzo(struct Persona *persona, const char *indirizzo);
void freePersona(struct Persona *persona);
```

persona.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define LEN_NOMINATIVO 30
#define LEN_INDIRIZZO 50
```

```
struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
};
```

```
struct Persona *newPersona(int id, char *nominativo, char *indirizzo) {
    struct Persona *persona = malloc(sizeof(struct Persona));
    if (persona == NULL) {
        printf("Errore nell'allocazione della persona\n");
        exit(1);
    }

    persona->id = id;
    strcpy(persona->nominativo, nominativo);
    strcpy(persona->indirizzo, indirizzo);
    return persona;
}
```

```
}

int getId(struct Persona *persona) { return persona->id; }

char *getNominativo(struct Persona *persona) { return persona->nominativo; }

char *getIndirizzo(struct Persona *persona) { return persona->indirizzo; }

bool setIndirizzo(struct Persona *persona, const char *indirizzo) {
    if (strcmp(indirizzo, "") != 0) {
        strcpy(persona->indirizzo, indirizzo);
        return true;
    }
    return false;
}

void freePersona(struct Persona *persona) {
    free(persona);
}
```

Una possibile esecuzione produce il seguente risultato.

```
Id: 1
Nominativo: Minni
Indirizzo: Via Torino, 6
1 - Minni, Via Torino, 6
1 - Minni, Via Roma, 1
```

Esempio con strutture nascoste e modificatore static

Il seguente esempio differisce dal precedente per il fatto che vengono gestite due persone distinte e l'impostazione del campo `id` non viene effettuata tramite una semplice assegnazione, bensì usando una variabile locale `static` all'interno della funzione `newPersona()`. La variabile `static` permette di mantenere la numerazione durante la creazione delle diverse persone, consentendo di assegnare automaticamente una numerazione incrementale. La funzione `newPersona()` necessita ora di un parametro formale in meno dato che l'id viene gestito automaticamente.

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include "lib/dimensioni/dimensioni.h"
#include "lib/persona/persona.h"

int main(void) {
    struct Persona *persona1;
    struct Persona *persona2;
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];

    // Caricamento dati per creare la 1^ persona
    printf("Nominativo: ");
    fgets(nominativo, LEN_NOMINATIVO, stdin);
    nominativo[strlen(nominativo) - 1] = '\0';
    printf("Indirizzo: ");
    fgets(indirizzo, LEN_INDIRIZZO, stdin);
    indirizzo[strlen(indirizzo) - 1] = '\0';

    // Creazione di una nuova persona
    persona1 = newPersona(nominativo, indirizzo);

    // Caricamento dati per creare la 2^ persona
    printf("Nominativo: ");
    fgets(nominativo, LEN_NOMINATIVO, stdin);
    nominativo[strlen(nominativo) - 1] = '\0';
    printf("Indirizzo: ");
    fgets(indirizzo, LEN_INDIRIZZO, stdin);
    indirizzo[strlen(indirizzo) - 1] = '\0';

    // Creazione di una nuova persona
    persona2 = newPersona(nominativo, indirizzo);

    // Visualizzazione dei dati delle due persone
    printf("Persona 1: ");
    printf("%d - %s, %s\n", getId(persona1), getNominativo(persona1),
        getIndirizzo(persona1));

    printf("Persona 2: ");
    printf("%d - %s, %s\n", getId(persona2), getNominativo(persona2),
        getIndirizzo(persona2));
}
```

```

// Modifica dell'indirizzo della persona
bool flag = setIndirizzo(&persona2, "Via Roma, 1");
if (!flag) printf("Indirizzo non modificato\n");

// Visualizzazione dei dati delle due persone
printf("Persona 1: ");
printf("%d - %s, %s\n", getId(persona1), getNominativo(persona1),
      getIndirizzo(persona1));

printf("Persona 2: ");
printf("%d - %s, %s\n", getId(persona2), getNominativo(persona2),
      getIndirizzo(persona2));

// Deallocazione delle persone
freePersona(persona1);
freePersona(persona2);

return 0;
}

```

Il file **dimensioni.h** contiene le macro per dimensionare il nominativo e l'indirizzo di una persona.

dimensioni.h

```

#pragma once

#define LEN_NOMINATIVO 30
#define LEN_INDIRIZZO 50

```

Il file **persona.h** rappresenta l'interfaccia di comunicazione per poter operare con una generica Persona. In esso è riportata la sola dichiarazione della struttura di una persona.

```
struct Persona;
```

Dato che la struttura è solo dichiarata, ma non definita, è necessario che vengano fornite le funzioni per agire sui campi della persona. Le seguenti funzioni permettono di accedere in lettura ai campi.

```

int getId(struct Persona *persona);
char *getNominativo(struct Persona *persona);
char *getIndirizzo(struct Persona *persona);

```

Dato che si è scelto di consentire la modifica del solo indirizzo, è presente la funzione che effettua tale modifica.

```
bool setIndirizzo(struct Persona *persona, const char *indirizzo);
```

persona.h

```

#pragma once

struct Persona; // Struttura nascosta, detto anche tipo incompleto

struct Persona *newPersona(char *nominativo, char *indirizzo);
int getId(struct Persona *persona);
char *getNominativo(struct Persona *persona);
char *getIndirizzo(struct Persona *persona);
bool setIndirizzo(struct Persona *persona, const char *indirizzo);
void freePersona(struct Persona *persona);

```

Nel file **persona.c** si trova la definizione della struttura che definisce una persona.

```
struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
};
```

Nel file **persona.c** la funzione `newPersona()` permette l'allocazione della singola persona tramite l'uso della funzione `malloc()`.

```
struct Persona *persona = malloc(sizeof(struct Persona));
```

Dato che la variabile **persona** è un puntatore, per accedere ai singoli campi della struttura è necessario usare l'operatore `->` (*arrow*).

```
persona->id = id;
strcpy(persona->nominativo, nominativo);
strcpy(persona->indirizzo, indirizzo);
```

Nel file sono anche definite tutte le funzioni necessarie per interagire con una persona.

persona.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "../dimensioni/dimensioni.h"

/**
 * Struttura che definisce una persona
 */
struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
};

/**
 * Crea una nuova persona.
 * L'identificatore viene generato automaticamente.
 * @param nominativo Nominativo della persona
 * @param indirizzo Indirizzo della persona
 * @return l'indirizzo nella memoria Heap dello spazio di memoria atto a contenere i
 * dati di una persona
 */
struct Persona *newPersona(char *nominativo, char *indirizzo) {
    static int id = 0;
    struct Persona *persona = malloc(sizeof(struct Persona));
    if (persona == NULL) {
        printf("Errore nell'allocazione della persona\n");
        exit(1);
    }
}
```

```
    persona->id = ++id;
    strcpy(persona->nominativo, nominativo);
    strcpy(persona->indirizzo, indirizzo);
    return persona;
}

/**
 * Restituisce l'identificatore di una persona.
 * @param persona Persona di cui restituire l'identificatore
 * @return l'identificatore della persona
 */
int getId(struct Persona *persona) { return persona->id; }

/**
 * Restituisce il nominativo di una persona.
 * @param persona Persona di cui restituire il nominativo
 * @return il puntatore alla stringa del nominativo
 */
char *getNominativo(struct Persona *persona) { return persona->nominativo; }

/**
 * Restituisce l'indirizzo di una persona.
 * @param persona Persona di cui restituire l'indirizzo
 * @return il puntatore alla stringa dell'indirizzo
 */
char *getIndirizzo(struct Persona *persona) { return persona->indirizzo; }

/**
 * Imposta l'indirizzo di una persona.
 * @param persona Persona di cui impostare l'indirizzo
 * @param indirizzo Indirizzo da impostare
 * @return true se la modifica avviene correttamente, false altrimenti
 */
bool setIndirizzo(struct Persona *persona, const char *indirizzo) {
    if (strcmp(indirizzo, "") != 0) {
        strcpy(persona->indirizzo, indirizzo);
        return true;
    }
    return false;
}

/**
 * Dealloca una persona.
 * @param persona Persona da deallocare
 */
void freePersona(struct Persona *persona) {
    free(persona);
}
```

Esercizi

157. Riprendi l'esercizio relativo alle città e al corrispondente PM10 e crea una struct in grado di gestire l'entità complessiva della città e della misurazione del PM10. La struct deve essere salvata nel file `citta.c` insieme alle funzioni che manipolano la singola entità. In questo file devi definire anche le funzioni per:

- a. allocare lo spazio nello Heap per una singola città con il suo PM10 e l'assegnazione dei valori;
- b. deallocare lo spazio nello Heap dell'entità;
- c. la funzione getter per il nome della città;
- d. la funzione getter per il valore del PM10;
- e. La funzione setter per il solo PM10;

Nel file `citta.h` devi esporre le funzioni e il tipo incompleto (struct nascosta) dell'entità. Usa l'allocazione dinamica della memoria e la programmazione modulare seguendo tutti i principi imparati finora.

Array dinamico di struct

C Tutorial

2. VIDEO - [Dynamically Allocate An Array Of Structs | C Programming Tutorial](#) by [Portfolio Courses](#)

Esempio di array dinamico di struct

In questa applicazione si vuole gestire l'elenco degli studenti di una scuola e, per farlo viene usato un array dinamico di struct.

La gestione di un array dinamico di struct, a loro volta dinamiche, non differisce dalla gestione di un array dinamico di stringhe dinamiche. In entrambi i casi è necessario creare un array di puntatori.

```
struct Persona **studenti;
```

Ciò che differisce è il tipo assegnato all'array che in questo caso è **struct Persona**.

Nella funzione main() viene richiesto l'inserimento del numero di studenti e si procede successivamente all'allocazione dell'array.

```
studenti = allocaArray(numStudenti);
```

Successivamente si procede con il caricamento dei dati.

```
caricaArray(studenti, numStudenti, "Inserisci i dati dello studente");
```

Con la visualizzazione dell'array è possibile verificare il corretto caricamento dei dati.

```
visualizzaArray(studenti, numStudenti);
```

main.c

```
/**
 * L'applicazione carica, ordina e visualizza una collezione di studenti
 */
#include <stdio.h>
#include <string.h>
#include "lib/persona/persona.h"
#include "lib/arraypersona/arraypersona.h"

int main(void) {
    struct Persona **studenti;
    int numStudenti;

    printf("Inserisci il numero di studenti: ");
    scanf("%d", &numStudenti);
    getchar();

    // Allocazione dell'array di studenti
    studenti = allocaArray(numStudenti);

    // Caricamento degli studenti
```

```
caricaArray(studenti, numStudenti, "Inserisci i dati dello studente");
// Visualizzazione degli studenti
visualizzaArray(studenti, numStudenti);
// Ordinamento degli studenti in base al nominativo
ordinaArray(studenti, numStudenti);
// Visualizzazione degli studenti ordinati
visualizzaArray(studenti, numStudenti);
// Dealloca l'array degli studenti
deallocaArray(studenti, numStudenti);
return 0;
}
```

Il file **dimensioni.h** contiene le macro per dimensionare il nominativo e l'indirizzo di una persona.

dimensioni.h

```
#pragma once
```

```
#define LEN_NOMINATIVO 20
#define LEN_INDIRIZZO 20
```

Il file **persona.h** rappresenta l'interfaccia di comunicazione per poter operare con una generica Persona. In esso è riportata la sola dichiarazione della struttura di una persona.

```
struct Persona;
```

Dato che la struttura è solo dichiarata, ma non definita, è necessario che vengano fornite le funzioni per agire sui campi della persona. Le seguenti funzioni permettono di accedere in lettura ai campi.

```
int getId(struct Persona *persona);
char *getNominativo(struct Persona *persona);
char *getIndirizzo(struct Persona *persona);
```

Dato che si è scelto di consentire la modifica del solo indirizzo, è presente la funzione che effettua tale modifica.

```
bool setIndirizzo(struct Persona *persona, const char *indirizzo);
```

persona.h

```
#pragma once
```

```
struct Persona; // Opaque structure with hidden members
```

```
struct Persona *newPersona(char *nominativo, char *indirizzo);
int getId(struct Persona *persona);
char *getNominativo(struct Persona *persona);
char *getIndirizzo(struct Persona *persona);
bool setIndirizzo(struct Persona *persona, const char *indirizzo);
void freePersona(struct Persona *persona);
```

Nel file **persona.c** si trova la definizione della struttura che definisce una persona.

```
struct Persona {
    int id;
    char nominativo[LEN_NOMINATIVO];
```

```
    char indirizzo[LEN_INDIRIZZO];  
};
```

Nel file **persona.c** la funzione `newPersona()` permette l'allocazione della singola persona tramite l'uso della funzione `malloc()`.

```
struct Persona *persona = malloc(sizeof(struct Persona));
```

Dato che la variabile **persona** è un puntatore, per accedere ai singoli campi della struttura è necessario usare l'operatore `->` (*arrow*).

```
persona->id = id;  
strcpy(persona->nominativo, nominativo);  
strcpy(persona->indirizzo, indirizzo);
```

Nel file sono anche definite tutte le funzioni necessarie per interagire con una persona.

persona.c

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdbool.h>  
#include "../dimensioni/dimensioni.h"  
  
/**  
 * Struttura che definisce una persona  
 */  
struct Persona {  
    int id;  
    char nominativo[LEN_NOMINATIVO];  
    char indirizzo[LEN_INDIRIZZO];  
};  
  
/**  
 * Crea una nuova persona.  
 * L'identificatore viene generato automaticamente.  
 * @param nominativo Nominativo della persona  
 * @param indirizzo Indirizzo della persona  
 * @return l'indirizzo nella memoria Heap dello spazio di memoria atto a contenere i  
 * dati di una persona  
 */  
struct Persona *newPersona(char *nominativo, char *indirizzo) {  
    static int id = 0;  
    struct Persona *persona = malloc(sizeof(struct Persona));  
    if (persona == NULL) {  
        printf("Errore nell'allocazione della persona\n");  
        exit(1);  
    }  
  
    persona->id = ++id;  
    strcpy(persona->nominativo, nominativo);  
    strcpy(persona->indirizzo, indirizzo);  
    return persona;  
}
```

```
/**
 * Restituisce l'identificatore di una persona.
 * @param persona Persona di cui restituire l'identificatore
 * @return l'identificatore della persona
 */
int getId(struct Persona *persona) { return persona->id; }

/**
 * Restituisce il nominativo di una persona.
 * @param persona Persona di cui restituire il nominativo
 * @return il puntatore alla stringa del nominativo
 */
char *getNominativo(struct Persona *persona) { return persona->nominativo; }

/**
 * Restituisce l'indirizzo di una persona.
 * @param persona Persona di cui restituire l'indirizzo
 * @return il puntatore alla stringa dell'indirizzo
 */
char *getIndirizzo(struct Persona *persona) { return persona->indirizzo; }

/**
 * Imposta l'indirizzo di una persona.
 * @param persona Persona di cui impostare l'indirizzo
 * @param indirizzo Indirizzo da impostare
 * @return true se la modifica avviene correttamente, false altrimenti
 */
bool setIndirizzo(struct Persona *persona, const char *indirizzo) {
    if (strcmp(indirizzo, "") != 0) {
        strcpy(persona->indirizzo, indirizzo);
        return true;
    }
    return false;
}

/**
 * Dealloca una persona.
 * @param persona Persona da deallocare
 */
void freePersona(struct Persona *persona) { free(persona); }
```

Il file **arraypersona.h** rappresenta l'interfaccia di comunicazione per poter operare con un array di persone. In esso sono presenti tutte le dichiarazioni delle funzioni che possono essere usate per interagire con un array di persone.

arraypersona.h

```
#pragma once
```

```
struct Persona **allocaArray(int numElementi);
void caricaArray(struct Persona **array, int numElementi, const char *messaggio);
void deallocaArray(struct Persona **array, int numElementi);
void visualizzaArray(struct Persona **array, int numElementi);
void ordinaArray(struct Persona **array, int numElementi);
```

Il modulo per la gestione dell'array di persone prevede la funzione per l'allocazione dell'array

```
struct Persona **allocaArray(int numElementi)
```

Che usa la funzione malloc() per creare l'array dinamico nello Heap il cui indirizzo viene memorizzato nello Stack Frame della funzione main() una volta che viene restituito dalla funzione allocaArray().

Il caricamento dell'array tramite la funzione

```
void caricaArray(struct Persona **array, int numElementi,
                 const char *messaggio)
```

prevede sia l'input dei dati necessari per la persona, sia l'invocazione della funzione di allocazione dinamica della singola persona, che restituisce un indirizzo di memoria nello Heap che viene memorizzato nell'attuale posizione dell'array.

```
array[i] = newPersona(id, nominativo, indirizzo);
```

La deallocazione dell'array deve prevedere sia la deallocazione delle singole persone, sia la deallocazione finale dell'intero array, questa è la ragione della presenza del ciclo nella funzione void deallocaArray(struct Persona **array, int numElementi). Il ciclo di deallocazione delle singole persone prevede l'invocazione della funzione freePersona(array[i]); per ogni persona presente nell'array (array[i]). Invocando questa funzione si applica il principio di delega che, appunto, delega al modulo di gestione della singola persona il compito di liberare la memoria Heap per la specifica persona.

La visualizzazione del contenuto dell'array viene gestito nella funzione

```
void visualizzaArray(struct Persona **array, int numElementi)
```

Tramite la funzione void ordinaArray(struct Persona **array, int numElementi) è possibile ordinare l'array in ordine crescente di nominativo della persona. Si osservi che per effettuare il confronto dei nominativi delle persone è necessario recuperarli tramite la funzione getNominativo() dato che la struct risulta nascosta.

arraypersona.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../dimensioni/dimensioni.h"
#include "../persona/persona.h"

/**
 * Alloca dinamicamente un array di puntatori.
 * @param numElementi - Numero di puntatori da allocare
 * @return l'indirizzo di memoria dell'array allocato nello Heap di puntatori ad una
 * collezione di persone
 */
struct Persona **allocaArray(int numElementi) {
    struct Persona **array;
    array = malloc(sizeof(struct Persona *) * numElementi);
    if (array == NULL) {
```

```
    printf("Errore nell'allocazione dell'array\n");
    exit(1);
}
return array;
}

/**
 * Carica l'array dinamico di elementi.
 * @param array - Array dinamico di elementi
 * @param lunElemento - Numero di byte che compongono un elemento
 * @param numElementi - Numero di elementi da caricare nell'array
 * @param messaggio - Messaggio da visualizzare per indicare cosa caricare
 */
void caricaArray(struct Persona **array, int numElementi,
                 const char *messaggio) {
    char nominativo[LEN_NOMINATIVO];
    char indirizzo[LEN_INDIRIZZO];
    // Caricamento delle persone
    for (int i = 0; i < numElementi; i++) {
        printf("%s %d:\n", messaggio, i + 1);
        // Caricamento dati per creare una nuova persona
        printf("Nominativo: ");
        fgets(nominativo, sizeof(nominativo), stdin);
        nominativo[strlen(nominativo) - 1] = '\0';
        printf("Indirizzo: ");
        fgets(indirizzo, LEN_INDIRIZZO, stdin);
        indirizzo[strlen(indirizzo) - 1] = '\0';
        array[i] = newPersona(nominativo, indirizzo);
    }
}

/**
 * Ordina l'array di stringhe seguendo un ordinamento lessicografico crescente.
 * @param array - Array da ordinare
 * @param numElementi - Numero di elementi presenti nell'array
 */
void ordinaArray(struct Persona **array, int numElementi) {
    for (int i = 0; i < numElementi - 1; i++) {
        for (int j = 0; j < numElementi - i - 1; j++) {
            if (strcmp(getNominativo(array[j]), getNominativo(array[j + 1])) > 0) {
                struct Persona *temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}

/**
 * Dealloca l'array di elementi deallocando prima ogni singolo elemento e poi
 * l'intero array.
 * @param array - Array da deallocare
 * @param numStringhe - Numero di elementi presenti nell'array
 */
void deallocaArray(struct Persona **array, int numElementi) {
```

```

    for (int i = 0; i < numElementi; i++) {
        freePersona(array[i]); // Principio di delega
    }
    free(array);
}

/**
 * Visualizza una linea decorativa parametrizzata
 * @param larghezzaNome - numero di "-" da visualizzare in base alla larghezza
 *                        del nome
 * @param larghezzaIndirizzo - numero di "-" da visualizzare in base alla
 *                        larghezza dell'indirizzo
 */
void stampaLinea(int larghezzaNome, int larghezzaIndirizzo) {
    printf("+----+");
    for (int i = 0; i < larghezzaNome; i++) printf("-");
    printf("+");
    for (int i = 0; i < larghezzaIndirizzo; i++) printf("-");
    printf("+\n");
}

/**
 * Visualizza elementi puntati dall'array.
 * @param array - Array dinamico di elementi
 * @param numElementi - Numero di elementi nell'array
 * @param messaggio - Messaggio da visualizzare per indicare cosa si sta
 *                  visualizzazando
 */
void visualizzaArray(struct Persona **array, int numElementi) {
    stampaLinea(LEN_NOMINATIVO, LEN_INDIRIZZO);
    printf("| Id |%-*s|%-*s|\n", LEN_NOMINATIVO, "Nome",
        LEN_INDIRIZZO, "Indirizzo");
    stampaLinea(LEN_NOMINATIVO, LEN_INDIRIZZO);

    for (int i = 0; i < numElementi; i++) {
        //
https://stackoverflow.com/questions/1809399/how-to-format-strings-using-printf-to-get-equal-length-in-the-output
        printf("| %4d |%-*s|%-*s|\n", getId(array[i]),
            LEN_NOMINATIVO, getNominativo(array[i]),
            LEN_INDIRIZZO, getIndirizzo(array[i]));
        stampaLinea(LEN_NOMINATIVO, LEN_INDIRIZZO);
    }
}

```

Una possibile esecuzione fornisce il seguente risultato.

```

Inserisci il numero di studenti: 3
Inserisci i dati dello studente 1:
Nominativo: Pippo Pippoli
Indirizzo: Via Tripoli, 8
Inserisci i dati dello studente 2:
Nominativo: Minni Minnoli
Indirizzo: Via Salerno, 5
Inserisci i dati dello studente 3:
Nominativo: Gastone Paperone

```

Indirizzo: Via Genova, 9

+-----+		+-----+	
Id	Nome	Address	
+-----+		+-----+	
1	Pippo Pippoli	Via Tripoli, 8	
+-----+		+-----+	
2	Minni Minnoli	Via Salerno, 5	
+-----+		+-----+	
3	Gastone Paperone	Via Genova, 9	
+-----+		+-----+	
+-----+		+-----+	
Id	Nome	Address	
+-----+		+-----+	
3	Gastone Paperone	Via Genova, 9	
+-----+		+-----+	
2	Minni Minnoli	Via Salerno, 5	
+-----+		+-----+	
1	Pippo Pippoli	Via Tripoli, 8	
+-----+		+-----+	

Esercizi

158. Riprendi l'esercizio relativo alle città e al corrispondente PM10 gestito tramite una struct usata per gestire oggetti dinamicamente nello Heap e, crea un array dinamico di elementi del tipo di questa struct.

La gestione della singola struct deve rimanere confinata nel file `citta.c` insieme alle funzioni che manipolano questa singola entità. In pratica nessuna variazione dovrebbe essere apportato a questo modulo, applicando il [principio Open/Closed](#)¹². Analogamente il file `citta.h` deve esporre le funzioni e il tipo incompleto (struct nascosta) dell'entità.

I singoli elementi che formalizzano una città con il corrispondente PM10 devono essere raccolti in un array dinamico di struct.

Come fatto nell'esercizio degli array dinamici paralleli, l'applicazione deve permettere di caricare i nomi delle città e del corrispondente valore del PM10, per un numero di città scelto dall'utente. Inoltre l'applicazione deve visualizzare la città con il PM10 massimo, la città con il PM10 minimo, la media di tutti i PM10 e l'elenco delle città con i corrispondenti PM10 in ordine decrescente di PM10. Aggiungi in questa applicazione anche la visualizzazione dei nomi delle città e del relativo PM10 in ordine crescente di nome città.

Usa l'allocazione dinamica della memoria e la programmazione modulare seguendo tutti i principi imparati finora.

159. Una biblioteca ti commissiona un'applicazione per la gestione dei suoi libri. Dai colloqui avvenuti con gli [stakeholder](#) è emerso che un aspetto importante è la gestione degli autori. Decidi quindi di partire con la progettazione di questo aspetto dell'applicazione. Per ogni autore bisogna mantenere un identificatore univoco che viene incrementato automaticamente dal sistema, il nominativo, il luogo di nascita, la data di nascita e l'eventuale luogo e data di morte. Nel caso di autori ancora viventi, questi due ultimi parametri devono essere inizializzati in modo opportuno (si scelga la modalità di inizializzazione).

Di tutti gli autori bisogna quindi prevedere il caricamento, la visualizzazione in ordine alfabetico di nominativo e la visualizzazione dei dati di un autore del quale è stato fornito il nominativo. Inizia a creare un menù temporaneo (diventerà un sottomenù) per gestire questi aspetti. Crea un modulo separato per la gestione del menù.

Crea inoltre un modulo che gestisce il singolo autore, nel quale prevedi la definizione della struct con le funzioni di allocazione e deallocazione del singolo autore, gestendo anche i getter e i setter che ritieni necessari.

Per la gestione dell'array di autori crea un modulo separato con tutte le funzioni necessarie per allocare e deallocare l'array e tutte le funzioni per svolgere le altre attività richieste sull'array. Questo modulo deve applicare il [principio di Delega](#)¹³, rimandando alle funzioni del modulo dell'autore la gestione degli aspetti del singolo autore.

Usa l'allocazione dinamica della memoria e la programmazione modulare seguendo tutti i principi imparati finora.

¹² In breve il principio Open/Closed afferma che, quando è possibile, il codice di un modulo non dovrebbe essere modificato, dovrebbe essere esteso inglobandolo all'interno di altri moduli che aggiungono nuove funzionalità.

¹³ In breve il principio di Delega prevede l'uso delle funzioni di un altro modulo che si occupa di gestire un'entità specifica, senza riscrivere nel nuovo modulo le funzioni richieste. Questo principio è fondamentale per progettare correttamente per moduli.

File

- autore.c → definizione struct Autore, costruttore (funzione con la malloc()), distruttore (funzione con la free()), getter (funzioni che restituiscono il valore di un campo della struct) e setter (funzioni che modificano il valore di un campo della struct);
- autore.h → dichiarazione struct Autore (struct nascosta o tipo incompleto), dichiarazione delle funzioni di autore.c;
- autori.c → #include "autore.h", costruttore (funzione con la malloc()), distruttore (funzione con il ciclo di free() per ogni autore e la free() dell'array), tutte le funzioni necessarie per gestire l'array;
- autori.h → dichiarazione delle funzioni di autori.c;
- menu.c → visualizzazione menù e inserimento scelta;
- menu.h → dichiarazione delle funzioni di menu.c;
- main.c → #include "menu.h", #include "autore.h", #include "autori.h", tutte le invocazioni necessarie.

160. Continuando con lo sviluppo dell'applicazione della biblioteca giungi alla fase di progettazione del singolo libro. Ogni libro deve presentare le seguenti caratteristiche: titolo, elenco degli autori, casa editrice, anno di pubblicazione, numero di pagine, codice identificativo del libro nella biblioteca.

Il codice identificativo del libro deve essere un valore autoincrementale gestito dal sistema che viene incrementato di 10 per ogni libro inserito.

L'elenco degli autori del libro deve essere gestito tramite un array dinamico di posizioni interno alla struct che modella il libro (*se non riesci a gestire un array dinamico interno alla struct, puoi usare un array dimensionato con una macro*). Nell'array degli autori del libro devono essere riportati i codici degli autori caricati nell'array degli autori.

Crea un modulo separato per la gestione del singolo libro che preveda la definizione della struct del libro, le funzioni di allocazione e deallocazione del libro e tutti i getter e i setter che ritieni siano utili. Decidi come gestire le funzioni per la gestione dell'array degli autori del libro, usando il più possibile i principi di Delega e Open/Closed.

Usa l'allocazione dinamica della memoria e la programmazione modulare seguendo tutti i principi imparati finora.

161. Finalmente giungi all'ultima fase del lavoro della biblioteca dove devi progettare ed implementare la collezione dei libri della biblioteca. Crea un modulo separato per la gestione della collezione ed usa i principi di Delega e Open/Closed per rendere modulare l'intera applicazione.

Il modulo che gestisce la biblioteca deve prevedere le funzioni per allocare e deallocare l'intero array di libri, includendo i getter e i setter che ritieni necessari.

Inoltre il modulo deve permettere di:

- a. caricare un libro;
- b. eliminare un libro;
- c. visualizzare tutti i dati di un libro cercato sia per titolo, sia per codice;
- d. elencare i libri in ordine crescente di titolo.

Crea un menù temporaneo (diventerà un sottomenù) per gestire questi aspetti. Usa il modulo del menù per la gestione degli autori per gestire anche questo menù.

Usa l'allocazione dinamica della memoria e la programmazione modulare seguendo tutti i principi imparati finora.

162. Per terminare il progetto della biblioteca prevedi la gestione di un menù principale che permetta di scegliere tra le seguenti voci: gestione autori, gestione libri, uscita. Scegliendo la gestione degli autori deve comparire il sottomenù che permette di caricare gli autori e di svolgere tutte le operazioni già viste durante lo sviluppo di questo modulo. Uscendo da questo sottomenù bisogna visualizzare nuovamente il menù principale. Se viene invece scelta la gestione dei libri deve comparire un sottomenù che permette di caricare i libri e svolgere tutte le operazioni già viste durante lo sviluppo di questo modulo. Uscendo da questo sottomenù bisogna visualizzare nuovamente il menù principale. La scelta di uscita deve invece prevedere la chiusura dell'applicazione. Usa il modulo di gestione dei menù che hai già creato per gestire i sottomenù degli autori e dei libri cercando di usare il più possibile la programmazione modulare, il riutilizzo del codice e i principi imparati finora.
163. La biblioteca sta cambiando sede, ma per completare il trasferimento ci vorranno almeno due anni. Nel frattempo i gestori della biblioteca non vogliono interrompere il servizio offerto ai cittadini e ti comunicano che la tua applicaizone sarà usata nelle due sedi della biblioteca fino al completamento del trasloco. Per i prossimi due anni la biblioteca varierà moltissimo la dimensione dell'array dei libri e quindi decidi di fornire un aggiornamento della tua applicazione che gestisca il ridimensionamento automatico dell'array dei libri. Questa modifica non avrà alcun impatto sui casi d'uso dell'applicazione in quanto è esclusivamente una gestione interna. Questo aggiornamento si occupa solo di aumentare la dimensione della collezione dei libri, per permettere di aggiungerne altri. La dimensione di un array viene modificata usando la funzione `realloc()` e si effettua il ridimensionamento quando l'array raggiunge la sua massima capacità, raddoppiando questo valore. Quindi, se l'array prevede 100 posizioni, una volta occupate tutte le posizioni, si rialloca lo spazio portandolo a 200 posizioni. Modifica la tua applicazione estendendo il modulo di gestione dell'array dei libri, prevedendo che questo venga ridimensionato adeguatamente quando necessario. Tale controllo deve avvenire ogni volta che un libro viene aggiunto.
164. In questo momento decidi di fornire un nuovo aggiornamento dell'applicazione di gestione della biblioteca... *riduzione dimensione*
- I limiti per decidere quando effettuare il ridimensionamento dell'array sono i seguenti:
- quando la dimensione attuale dell'array raggiunge la sua massima capacità, l'array deve essere ridimensionato raddoppiando l'attuale capacità massima;
 - quando la dimensione attuale dell'array raggiunge $\frac{1}{4}$ della sua massima capacità, l'array deve essere ridimensionato riducendo della metà la sua attuale capacità massima.
- Modifica la tua applicazione estendendo il modulo di gestione dell'array dei libri, prevedendo che questo venga ridimensionato adeguatamente se necessario. Tale controllo deve avvenire ogni volta che un libro viene aggiunto o eliminato dalla collezione. L'aumento della dimensione dell'array è legato all'aggiunta dei libri, attività che hai gestito finora.

La diminuzione della dimensione dell'array è invece collegata all'eliminazione di libri dall'array. In questo caso devi ricordare che se un libro viene eliminato, deve anche essere deallocato dalla memoria Heap e, se la sua posizione non è l'ultima dell'array, la collezione deve essere compattata.

165. *Da scrivere. Modifica il codice del libro in modo da assegnare una numerazione seguendo la [Classificazione decimale Dewey](#).*

Puntatore a void [approfondimento]

C Tutorial

31. [Void Pointer in C](#) by Ankit Kochar, [PrepBytes Blog](#)
32. VIDEO by [Portfolio Courses](#) in [C Programming Tutorials](#)
 - a. [Void Pointers](#)

Esempio su puntatori a void e allocazione dinamica

La seguente applicazione ripropone lo scambio di due elementi, ma in questo caso il tipo degli elementi scambiati non è conosciuto dalla funzione che effettua lo scambio.

Per creare una **funzione di scambio generica** è necessario utilizzare come **parametri formali dei puntatori a void**. Questo permette di generalizzare la funzione di scambio e renderla utilizzabile per qualsiasi tipo di elemento da scambiare, **rendendo effettivamente modulare ed indipendente la libreria**.

Dato che un puntatore a void deve essere tipizzato, la funzione di scambio necessita di un ulteriore parametro che corrisponde alla dimensione del tipo che si vuole utilizzare.

```
void scambio(void *first, void *second, size_t size)
```

Il tipo `size_t` è un unsigned int definito all'interno della libreria `stdio.h` che contiene la dimensione del tipo che si desidera utilizzare e corrisponde al valore restituito dalla funzione `sizeof()`.

La funzione `scambio()` deve utilizzare al suo interno la funzione `malloc()` per allocare lo spazio di memoria necessario per la variabile locale `temp`. Inoltre deve usare la funzione `memcpy()` che effettua la copia del contenuto di un blocco di memoria in un altro blocco di memoria per la dimensione indicata. Per la precisione, il prototipo della funzione `memcpy()` è il seguente:

```
void * memcpy ( void * destination, const void * source, size_t num );
```

La funzione copia un blocco di memoria di dimensione `num`, da `source` a `destination`. Il contenuto puntato dal parametro formale `source` viene considerato immutabile per la presenza del modificatore `const`.

Questo semplice esempio mostra come l'utilizzo di **parametri formali di tipo void*** permetta di **generalizzare** il comportamento delle **funzioni consentendo il loro uso con qualsiasi tipo di parametro attuale**.

main.c

```
/**
 * Dati due elementi li scambia di posizione
 * @author mgm
 */
#include <stdlib.h>
#include <stdio.h>
#include "lib/utility.h"

int main() {
    int primoInt, secondoInt;;
    double primoDouble, secondoDouble;
    char primoChar, secondoChar;
    // ---- INT
    // Inserimento numeri interi
    printf("Inserire il 1° intero: ");
    scanf("%d", &primoInt);
    printf("Inserire il 2° intero: ");
    scanf("%d", &secondoInt);

    // Scambio i numeri interi
    scambio(&primoInt, &secondoInt, sizeof(int));

    // Visualizzo i numeri scambiati
    printf("1° intero: %d\n", primoInt);
    printf("2° intero: %d\n", secondoInt);
    // ---- DOUBLE
    // Inserimento numeri double
    printf("Inserire il 1° double: ");
    scanf("%lf", &primoDouble);
    printf("Inserire il 2° double: ");
    scanf("%lf", &secondoDouble);

    // Scambio i numeri double
    scambio(&primoDouble, &secondoDouble, sizeof(double));

    // Visualizzo i numeri scambiati
    printf("1° double: %lf\n", primoDouble);
    printf("2° double: %lf\n", secondoDouble);
    // ---- CHAR
    // Inserimento caratteri
    printf("Inserire il 1° carattere: ");
    getchar();
    scanf("%c", &primoChar);
    printf("Inserire il 2° carattere: ");
    getchar();
    scanf("%c", &secondoChar);

    // Scambio i caratteri
    scambio(&primoChar, &secondoChar, sizeof(char));

    // Visualizzo i caratteri scambiati
    printf("1° carattere: %c\n", primoChar);
    printf("2° carattere: %c\n", secondoChar);
```

```
    return 0;
}
```

utility.h

```
#ifndef UTILITY_H
#define UTILITY_H
#include <stdio.h>
```

```
void scambio(void* first, void* second, size_t size);
```

```
#endif
```

utility.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
void scambio(void *first, void *second, size_t size) {
    // Alloco la memoria necessaria
    void *temp = malloc(size);

    // Scambio i valori
    memcpy(temp, first, size);
    memcpy(first, second, size);
    memcpy(second, temp, size);

    // Dealloco la memoria
    free(temp);
}
```

File

C Tutorial

Guarda i seguenti video per comprendere il funzionamento dei file di testo e i file binari.

33.VIDEO by [Portfolio Courses](#)

- a. File di testo
 - i. [File Access Basics | C Programming Tutorial](#)
 - ii. [Read CSV File Data Into An Array Of Structs | C Programming Example](#)
 - iii. [Write An Array Of Structs To A CSV File | C Programming Example](#)
 - iv. [rewind\(\) function to reset file pointer to file start | C Programming Tutorial](#)
- b. File binario ← **facoltativo**
 - i. [Binary File Access Introduction | C Programming Example](#)
 - ii. [Read And Write An Array Of Structs To A Binary File | C Programming Example](#)

Esempi

File access basics

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int main() {
    FILE *fileOut;

    fileOut = fopen("io.txt", "w");
    /*fputs("abc", file);
    fputs("123\n", file);
    fputs("A string\n", file);
    int data = 5;
    fprintf(file, "data: %d\n", data);*/

    /*for(int i = 0; i < 10; i++) {
        fprintf(file, "%d\n", i);
    }*/

    int input = 0;
    do {
        printf("Enter # (-1 quits): ");
        scanf("%d", &input);
        if(input != -1) fprintf(fileOut, "%d\n", input);
    } while(input != -1);
    fclose(fileOut);

    FILE *fileIn;
    fileIn = fopen("io.txt", "r");
    int finput = 0;
    int lines = 0;
```

```
int numbers[100];

/*while(fscanf(fileIn, "%d", &finput) != EOF) {
    numbers[lines++] = finput;
    printf("File line %d: %d\n",finput);
}*/

do {
    int read = fscanf(fileIn, "%d", &finput);
    if(read != EOF) {
        numbers[lines++] = finput;
        printf("File line %d: %d\n",lines, finput);
    }
} while(!feof(fileIn));

fclose(fileIn);

fileIn = fopen("in.txt", "r");

char buffer[4096];
fgets(buffer, 4096, fileIn);
printf("Buffer: %s\n", buffer);

fclose(fileIn);
return 0;
}
```

Read CSV file data into an array of structs

```
#include <stdio.h>
#include <stdlib.h>

struct Student {
    char type;
    char name[50];
    int age;
    double average;
};

int main() {
    // funzione FILE *apri(const char *nomeFile, const char *modalità);
    FILE *file;
    file = fopen("file.txt", "r");
    if(file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    // end funzione

    // funzione struct Student *leggi(FILE *file);
    struct Student students[100];
    int read = 0;
    int records = 0;
    do {
        /*
        * La stringa "%c#,%49[^#]#%d#%lf\n" indica:
        * %c# --> leggi un carattere (campo type) e poi il simbolo '#'
        * %49[^#]# --> leggi 49 caratteri (name), ma smetti appena trovi # ([^#]),
        *             ma subito dopo deve trovare il simbolo '#'
        * %d# --> legge un intero seguito dal carattere '#'
        * %lf --> legge un double
        * \n --> legge un new line
        */
        read = fscanf(file,
                     "%c#,%49[^#]#%d#%lf\n",
                     &students[records].type,
                     students[records].name,
                     &students[records].age,
                     &students[records].average);
        if(read == 4) records++;
        if(read != 4 && !feof(file)) {
            printf("File format incorrect.\n");
            return 1;
        }
        if(ferror(file)) {
            printf("Error reading file.\n");
            return 1;
        }
    } while(!feof(file));
    // end funzione

    // funzione void visualizza(struct Student *students,
```

```
//                                const char *format,
//                                const char *msg);
printf("\n%d records read.\n\n", records);
for(int i = 0; i < records; i++) {
    printf("%c, %s, %d, %.2lf\n",
           students[i].type,
           students[i].name,
           students[i].age,
           students[i].average);
}
printf("\n");
// end funzione
return 0;
}
```

Esercizi

166. L'Istat, l'Istituto Nazionale di Statistica, commissiona alla società per cui lavori un aggiornamento del software di gestione del censimento della popolazione italiana, e tu vieni incaricato di sviluppare una porzione del software.

Nello specifico ti devi occupare di un software in grado di manipolare i nomi delle Regioni d'Italia con la relativa densità di abitanti per km².

Le 20 Regioni d'Italia con la corrispondente densità abitativa sono: Friuli-Venezia Giulia - 151, Basilicata - 53, Sardegna - 65, Emilia-Romagna - 197, Valle d'Aosta - 38, Lombardia - 417, Molise - 65, Lazio - 331, Toscana - 159, Puglia - 200, Campania - 410, Sicilia - 186, Trentino-Alto Adige - 79, Abruzzo - 117, Umbria - 101, Marche - 157, Piemonte - 167, Veneto - 264, Liguria - 278, Calabria - 121.

La tua applicazione deve occuparsi di svolgere le seguenti operazioni permettendo di sceglierle tramite un menù. Il menù deve anche permettere di chiudere l'applicazione.

- a. Caricamento delle regioni con la relativa densità di abitanti. Il caricamento deve avvenire in modo che i nomi delle regioni risultino sempre ordinate. Una volta caricate in memoria le regioni, queste dovranno essere salvate in un file di testo in modo tale che i dati possano essere caricati in memoria nelle successive esecuzioni del programma.
- b. Visualizzazione dell'elenco delle regioni e della relativa densità di abitanti.
- c. Visualizzazione della densità di abitanti di una regione data.
- d. Modifica della densità di abitanti di una regione data. Questa scelta deve implicare un nuovo salvataggio dei dati che vanno a sostituire i precedenti.

Usa la programmazione modulare e le strutture dati dinamiche. Per l'ordinamento usa un algoritmo tra quelli visti a lezione. Come algoritmo di ricerca usa l'algoritmo di ricerca binario. Memorizza le struct in un file di testo come indicato nel video [Write An Array Of Structs To A CSV File | C Programming Example](#) e leggi i dati usando la modalità indicata nel video [Read CSV File Data Into An Array Of Structs | C Programming Example](#).

167. Progettare un algoritmo che permetta di caricare e salvare su file binario l'elenco dei docenti dell'Istituto con la classe di concorso e il relativo punteggio in graduatoria.

L'algoritmo dovrà permettere di effettuare le seguenti operazioni:

- a. visualizzare l'elenco di tutti i docenti in ordine crescente di cognome;
- b. visualizzare le graduatorie per ogni singola classe di concorso, considerando che le graduatorie vengono visualizzate in ordine decrescente di punteggio;
- c. effettuare la ricerca di un docente per visualizzare la sua classe di concorso e il punteggio;
- d. modificare il punteggio del docente.

Usare la programmazione modulare e le strutture dati dinamiche. Per l'ordinamento si usi un algoritmo tra quelli visti a lezione. Come algoritmo di ricerca si usi l'algoritmo di ricerca binario. Memorizza e leggi i dati delle struct in un file binario come indicato nel video [Read And Write An Array Of Structs To A Binary File | C Programming Example](#).

168. bla bla

Algoritmi

Algoritmi di ordinamento

Bubble sort

Read the following article by about **Bubble sort** in order to grasp the main concept of this algorithm.

- [Bubble Sort Algorithm](#) using number (**you have to read it!**) by tutorialspoint.com
- [C Programming: Sorts the strings of an array using bubble sort](#) using string (**you have to read it!**) by w3resource

Read carefully the solution given in C in order to understand how it works.

Watch the following video by [Portfolio Courses](#)

- [Bubble Sort | C Programming Example](#) (**you have to watch it!**)
- [Selection Sort | C Programming Example](#) (*optional*)
- [Insertion Sort | C Programming Example](#) (*optional*)
- [Merge Sort | C Programming Example](#) (*optional*)
- [Quicksort Algorithm Implementation | C Programming Example](#) (*optional*)

Watch also the following videos by [Yusuf Shakeel](#). Some of them are without audio.

- [Sorting Algorithm | Bubble Sort - step by step guide](#) (**you have to watch it!**)
- [Sorting Algorithm | Selection Sort - step by step guide](#) (*optional*)
- [Sorting Algorithm | Insertion Sort - step by step guide](#) (*optional*)
- [Sorting Algorithm | Merge Sort - step by step guide](#) (*optional*)
- [Sorting Algorithm | Quick Sort - step by step guide](#) (*optional*)

Esercizi

169. Progettare un'applicazione che permette di gestire le medie finali di una classe. Della classe si vogliono caricare i nomi degli studenti e la media finale per ogni studente. Una volta caricati i dati l'applicazione deve fornire la possibilità di visualizzare l'elenco di tutti gli studenti con le relative medie. Inoltre si vuole ottenere a video l'elenco degli studenti con la media migliore e l'elenco di quelli con la media peggiore. Infine l'applicazione deve permettere di visualizzare l'elenco degli studenti con le loro medie con i seguenti ordinamenti:

- a. in ordine crescente di nome studente;
- b. in ordine decrescente di voto.

Si usi un menù per gestire le diverse opzioni richieste. Si faccia riferimento all'esempio [Array paralleli stringhe con elenco di singoli numeri](#) per capire come gestirli.

Presentare l'analisi e la codifica dell'applicazione.

170. Progettare un'applicazione in grado di prendere in input dei valori numerici interi, allocandoli in un array e li visualizzi secondo le seguenti modalità; negli esempi si suppone che il contenuto dell'array sia 3, 5, 2, 7, 8, 23, 12, 34, 9. Si usi un menù per scegliere il tipo di attraversamento dell'array.

- a. Attraversamento lineare, cioè si attraversa la collezione dal primo elemento all'ultimo elemento, quindi deve essere visualizzata la sequenza 3, 5, 2, 7, 8, 23, 12, 34, 9.
- b. Attraversamento sinistra-destra, cioè si attraversa la collezione partendo dal primo elemento, poi si passa all'ultimo, poi al secondo, poi al penultimo e così via e, la sequenza visualizzata è 3, 9, 5, 34, 2, 12, 7, 23, 8.
- c. Attraversamento destra-sinistra, cioè si attraversa la collezione partendo dall'ultimo elemento, poi si passa al primo, poi si passa al penultimo, poi al secondo e così via e, la sequenza visualizzata è 9, 3, 34, 5, 12, 2, 23, 7, 8.
- d. Attraversamento inverso, cioè si attraversa la collezione dall'ultimo elemento al primo elemento, visualizzando la sequenza 9, 34, 12, 23, 8, 7, 2, 5, 3.
- e. Attraversamento ordinato crescente, cioè si ordina la collezione in ordine crescente e la si visualizza ottenendo 2, 3, 5, 7, 8, 9, 12, 23, 34.
- f. Attraversamento ordinato decrescente, cioè si ordina la collezione in ordine decrescente e la si visualizza ottenendo 34, 23, 12, 9, 8, 7, 5, 3, 2.

Usare la programmazione modulare prevedendo quanto segue.

- Un modulo che contenga tutte le funzioni per la gestione degli array.
- Un modulo che contenga tutte le funzioni di attraversamento dell'array.

Presentare l'analisi e la codifica dell'applicazione.

Algoritmi di ricerca

Binary search

Read the following articles by [Khan Academy](#) about **Binary search** in order to grasp the main concept of this algorithm.

1. [Binary search](#)
2. [Implementing binary search of an array](#)
3. [Challenge: Binary search](#) - Try it in C and Python language. Don't use any AI to do this exercise, the AI already knows how to do it, instead you need to learn it!
4. [Running time of binary search](#)
5. [Running time of binary search - Questions](#)

Exercises

171. Create a C application using the indications that are provided in the 3rd point of the previous paragraph.
172. Create a C application to find a given name in an ordered string array.

Esempi di ricerca binaria

Memoria statica

...

Memoria dinamica

Nel seguente esempio viene utilizzato il programma di gestione di un Array di persone per gestire l'elenco di un gruppo di studenti.

Nel modulo di gestione dell'array è stata aggiunta la funzione di ricerca binaria che permette di cercare una persona in base a nominativo

```
struct Persona *ricercaElemento(struct Persona **array, int numElementi, char *criterio)
```

Di seguito viene riportata la funzione main.c dove è stata aggiunta la richiesta del nominativo da cercare con la successiva visualizzazione dei dati.

Sono inoltre presenti i file arraypersona.h e una porzione di arraypersona.c nei quali è stata aggiunta la funzione di ricerca.

Il resto dell'applicazione rimane invariato rispetto al precedente esempio e quindi non viene riportato.

main.c

```
/**
 * L'applicazione carica, ordina e visualizza una collezione di studenti
 */
#include "lib/arraypersona/arraypersona.h"
#include "lib/dimensioni/dimensioni.h"
#include "lib/persona/persona.h"
#include <stdio.h>
#include <string.h>

int main(void) {
    struct Persona **studenti;
    int numStudenti;

    printf("Inserisci il numero di studenti: ");
    scanf("%d", &numStudenti);
    getchar();

    // Allocazione dell'array di studenti
    studenti = allocaArray(numStudenti);

    // Caricamento degli studenti
    caricaArray(studenti, numStudenti, "Inserisci i dati dello studente");
    // Visualizzazione degli studenti
```

```
visualizzaArray(studenti, numStudenti);
// Ordinamento degli studenti in base al nominativo
ordinaArray(studenti, numStudenti);
// Visualizzazione degli studenti ordinati
visualizzaArray(studenti, numStudenti);
// Ricerca di uno studente in base al nominativo
char nominativo[LEN_NOMINATIVO];
printf("Inserisci il nominativo dello studente da cercare: ");
fgets(nominativo, LEN_NOMINATIVO, stdin);
nominativo[strlen(nominativo) - 1] = '\0';
struct Persona *studente = ricercaElemento(studenti, numStudenti,
nominativo);
if (studente != NULL) {
    printf("Studente trovato:\n");
    printf("%d - %s - %s\n", getId(studente), getNominativo(studente),
        getIndirizzo(studente));
} else {
    printf("Studente non trovato.\n");
}
return 0;
}
```

arraypersona.h

```
#pragma once
```

```
extern struct Persona **allocaArray(int numStringhe);
extern void caricaArray(struct Persona **array, int numStringhe, const char
*messaggio);
extern void deallocaArray(struct Persona **array, int numElementi);
extern void visualizzaArray(struct Persona **array, int numElementi);
extern void ordinaArray(struct Persona **array, int numElementi);
extern struct Persona *ricercaElemento(struct Persona **array, int
numElementi, char *criterio);
```

arraypersona.c

```
...
```

```
/**
 * Ricerca un elemento nell'array.
 * Viene usata la ricerca binaria e quindi l'array deve essere ordinato.
 * @param array Array di elementi
 * @param numElementi Numero di elementi presenti nell'array
 * @param criterio Elemento da cercare
 */
struct Persona *ricercaElemento(struct Persona **array, int numElementi, char
*criterio) {
    // Imposto l'indice min e max ai valori iniziali
    int min = 0;
    int max = numElementi - 1;
    // Avvio il ciclo di ricerca binaria
    do {
```

```
// Imposto l'indice guess al valore medio tra min e max
int guess = (min + max) / 2;
// Se il nominativo della persona puntata da guess è uguale al criterio...
if (strcmp(getNominativo(array[guess]), criterio) == 0) {
    // ...ritorno la persona puntata da guess
    return array[guess];
} else {
    // Se il nominativo della persona puntata da guess è minore del
    // criterio...
    if (strcmp(getNominativo(array[guess]), criterio) < 0) {
        // ...imposto l'indice min al valore successivo a quello di guess
        min = guess + 1;
    } else {
        // altrimenti, se il nominativo della persona puntata da guess è
        // maggiore del criterio imposto l'indice max al valore precedente a
        // quello di guess
        max = guess - 1;
    }
}
} while (min <= max); // Ciclo fino a quando min <= max.
// Se min diventa maggiore di max il nominativo non esiste e restituisco NULL
return NULL;
}
```

Algoritmi ricorsivi

Recursion

Read the following articles by [Khan Academy](#) about **Recursion** in order to grasp the main concept of this programming technique.

1. [Recursion](#)
2. [The factorial function](#)
3. [Challenge: Iterative factorial](#) - Try it in C language. Don't use any AI to do this exercise, the AI already knows how to do it, instead you need to learn it!
4. [Recursive factorial](#)
5. [Challenge: Recursive factorial](#) - Try it in C language. Don't use any AI to do this exercise, the AI already knows how to do it, instead you need to learn it!
6. ...

Recursive binary search

Watch the following video by [Portfolio Courses](#) for a recursive solution of the Binary Search Algorithm.

- [Binary Search Algorithm | C Programming Example](#)

Una questione di stile

Lo stile del codice è importante, fa la differenza tra un programma leggibile da chiunque e un programma incomprensibile. L'affermazione *leggibile da chiunque* include anche lo stesso autore del codice che solitamente dimentica dopo poco la ragione dell'esistenza di qualsiasi porzione di codice che non è assolutamente chiara, non commentata e che non segue alcuna buona pratica di programmazione.

Per questa ragione consiglio di imparare lo stile dei linguaggi che userete, vi renderà dei programmatori migliori e, soprattutto, vi farà risparmiare tanto tempo e tanta fatica.

Di seguito vengono consigliate alcune letture.

- [Linux kernel coding style](#)
- The Pragmatic Programmer, 20th Anniversary Edition by David Thomas and Andrew Hunt
- Code Complete: A Practical Handbook of Software Construction, Second Edition by Steve McConnell
- Clean Code: A Handbook of Agile Software Craftsmanship by Martin Robert C.
- ... e [altri](#).

Live long and prosper, keep calm and enjoy coding, may we meet again.

Esercizi avanzati

Array

173. Progettare ed implementare un'applicazione in grado di calcolare e visualizzare l'indice di massa corporea di un gruppo di persone delle quali si conosce solo il peso e l'altezza. Usare tre array dinamici per memorizzare l'altezza, il peso e l'indice di massa corporea calcolato usando i dati dei due array precedenti. Visualizzare infine il contenuto dell'array degli indici di massa corporea. Usare la programmazione modulare per la gestione dell'applicazione creando i seguenti moduli:

- il modulo `imc` con una funzione per il calcolo dell'imc e una funzione per la visualizzazione dell'imc;
- Il modulo `gestionearray` con le funzioni per l'allocazione dinamica di un array di float e la funzione per la sua deallocazione.

main.c (*Chatraoui*)

```
/*Progettare ed implementare un'applicazione in grado di calcolare e visualizzare
 *l'indice di massa corporea di un gruppo di persone delle quali si conosce solo il
 *peso e l'altezza. Usare tre array dinamici per memorizzare l'altezza, il peso e
 *l'indice di massa corporea calcolato usando i dati dei due array precedenti.
 *Visualizzare infine il contenuto dell'array degli indici di massa corporea. Usare
 *la programmazione modulare per la gestione dell'applicazione.
 *@author Chatraoui Imad
 */
#include <stdio.h>
#include <float.h>
#include "lib/gestionearray/gestionearray.h"
#include "lib/imc/imc.h"

int main() {
    int numpersone;
    float* arraypeso;
    float* arrayaltezza;
    float* arrayimc;

    numpersone = dimensioneArray("Inserisci il numero di persone di cui vuoi calcolare
l'IMC: ");

    arraypeso = allocaArray(numpersone);
    arrayaltezza = allocaArray(numpersone);
    arrayimc = allocaArray(numpersone);

    caricaArray(arraypeso, numpersone, 1, FLT_MAX, "Peso della persona ");
    caricaArray(arrayaltezza, numpersone, -FLT_MAX, 3, "Altezza della persona ");

    imc(arraypeso, arrayaltezza, arrayimc, numpersone);

    visualizzaArray(arrayimc, numpersone, "IMC: ");
```

```
deallocaArray(arraypeso);
deallocaArray(arrayaltezza);
deallocaArray(arrayimc);
return 0;
}
```

gestionearray.h

```
#pragma once
extern float* allocaArray(int numElementi);
extern int dimensioneArray(const char* messaggio);
extern void deallocaArray(float* array);
extern void caricaArray(float* array, int numElementi, float limiteMin, float
limiteMax, const char* messaggio);
extern void visualizzaArray(float* array, int numElementi, const char* messaggio);
```

gestionearray.c

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
```

```
float* allocaArray(int numElementi) {
    float* array = (float *)malloc(numElementi * sizeof(float));

    if (array == NULL) {
        printf("Errore, memoria non disponibile");
        exit(1);
    }
    return array;
}
```

```
int dimensioneArray(const char* messaggio) {
    int numElementi;
    do {
        printf("%s", messaggio);
        scanf("%d", &numElementi);

        if (numElementi <= 0) {
            printf("La dimensione deve essere maggiore di 0.\n");
        }

    } while (numElementi <= 0);
    return numElementi;
}
```

```
void deallocaArray(float* array){
    free(array);
}
```

```
void caricaArray(float* array, int numElementi, float limiteMin, float limiteMax,
```

```
const char* messaggio) {

    for(int i = 0; i < numElementi; i++) {

        do {
            printf("%s %d: ", messaggio, i + 1);
            scanf("%f", &array[i]);

            if (array[i] < limiteMin || array[i] > limiteMax) {
                printf("Errore\n");
            }

        } while (array[i] < limiteMin || array[i] > limiteMax);
    }
}

void visualizzaArray(float* array, int numElementi, const char* messaggio) {
    for(int i = 0; i < numElementi; i++) {
        printf("%d) %s %f\n", i+1, messaggio, array[i]);
    }
}
```

imc.h

```
#pragma once
extern void imc(float* arraypeso, float* arrayaltezza, float* arrayimc, int
numpeople);
```

imc.c

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

void imc(float* arraypeso, float* arrayaltezza, float* arrayimc, int numpeople) {
    for(int i = 0; i < numpeople; i++) {
        arrayimc[i] = arraypeso[i] / (arrayaltezza[i] * arrayaltezza[i]);
    }
}
```

174. Progettare un algoritmo in grado di svolgere le quattro operazioni fondamentali su due numeri, estendendo il precedente esercizio.

Visualizzare inizialmente il seguente menù:

- 1) Somma
- 2) Sottrai
- 3) Moltiplica
- 4) Dividi
- 5) Esci

Se viene scelta una delle opzioni da 1 a 4 viene svolta l'operazione aritmetica richiesta sui due numeri visualizzando il risultato. Successivamente viene nuovamente visualizzato il menù e richiesto l'inserimento di due numeri. Se invece viene scelta l'opzione 5 il programma deve terminare, salutando cortesemente l'utente. Se viene inserito un valore non valido per il menù si deve riproporre il menù stesso.

Il progetto i precedenti e deve prevedere un modulo distinto per la gestione delle quattro operazioni aritmetiche usando delle funzioni che siano indipendenti dal tipo dei parametri forniti ([puntatore a void](#)), un modulo per la gestione dell'input e dell'output e un modulo per la gestione generica di un menù che preveda una funzione che accetti come parametro una stringa costante corrispondente al menù e restituisca la scelta fatta.

175. Progettare un'applicazione che permetta di caricare dei valori in un array, ma verifichi la presenza di ripetizioni. Se un valore fosse già presente l'algoritmo dovrà visualizzare "Presente", se non ci sono ripetizioni l'algoritmo dovrà visualizzare "Non presente". **La visualizzazione della stringa richiesta deve essere gestita nel main().** Usare l'allocazione dinamica della memoria per gestire l'array.

L'applicazione deve usare la programmazione modulare nel seguente modo.

- Il modulo gestionearray con le funzioni per l'allocazione dinamica di un array del tipo desiderato e la funzione per la sua deallocazione. Se il modulo è già stato creato per esercizi precedenti può essere riutilizzato.
- Aggiungere nel modulo gestionearray una funzione che cerca un valore all'interno di un generico array e restituisca un valore booleano opportuno per indicarne la presenza o l'assenza.

176. Progettare un algoritmo che implementi il gioco dell'impiccato. Si preveda un dizionario di parole precedentemente salvato in una struttura dati. L'algoritmo dovrà selezionare in modo pseudo-casuale una parola dal dizionario e usarla per il gioco. Per ogni tentativo fallito si visualizzi una frase che indichi quali sono le parti del corpo già perdute (testa, corpo, braccio destro, braccio sinistro, gamba destra, gamba sinistra). Per ogni tentativo dovrà essere visualizzata la sequenza di caratteri indovinata nella giusta posizione e tutta la sequenza di caratteri usata fino a quel momento. L'utente dovrà inserire un carattere per volta come tentativo per completare la parola da indovinare. Si utilizzi la programmazione modulare per implementare il gioco.

Algoritmi di ordinamento

177. Progettare un'applicazione in grado di prendere in input dei valori numerici interi, allocandoli in un array dinamico e, li visualizzi secondo le seguenti modalità; negli esempi si suppone che il contenuto dell'array sia 3, 5, 2, 7, 8, 23, 12, 34, 9.

- a. Attraversamento lineare, cioè si attraversa la collezione dal primo elemento all'ultimo elemento, quindi deve essere visualizzata la sequenza 3, 5, 2, 7, 8, 23, 12, 34, 9.
- b. Attraversamento sinistra-destra, cioè si attraversa la collezione partendo dal primo elemento, poi si passa all'ultimo, poi al secondo, poi al penultimo e così via e, la sequenza visualizzata è 3, 9, 5, 34, 2, 12, 7, 23, 8.
- c. Attraversamento destra-sinistra, cioè si attraversa la collezione partendo dall'ultimo elemento, poi si passa al primo, poi si passa al penultimo, poi al secondo e così via e, la sequenza visualizzata è 9, 3, 34, 5, 12, 2, 23, 7, 8.
- d. Attraversamento inverso, cioè si attraversa la collezione dall'ultimo elemento al primo elemento, visualizzando la sequenza 9, 34, 12, 23, 8, 7, 2, 5, 3.

- e. Attraversamento ordinato crescente, cioè si ordina la collezione in ordine crescente e la si visualizza ottenendo 2, 3, 5, 7, 8, 9, 12, 23, 34.
- f. Attraversamento ordinato decrescente, cioè si ordina la collezione in ordine decrescente e la si visualizza ottenendo 34, 23, 12, 9, 8, 7, 5, 3, 2.

Usare la programmazione modulare prevedendo quanto segue.

- Un modulo che contenga tutte le funzioni per la gestione degli array, incluse quelle per l'allocazione dinamica di un array e la sua deallocazione. Se questo modulo è già stato creato per altre applicazioni può essere riutilizzato.
- Un modulo che contenga tutte le funzioni di attraversamento dell'array.

La seguente soluzione non è del tutto corretta per i punti (b) e (c) e, mancano i punti (e) ed (f). Cercate di risolvere gli errori e completate l'esercizio.

178. Progettare un'applicazione in grado di prendere in input un valore numerico intero e visualizzi delle figure geometriche, usando delle funzioni che restituiscono le seguenti stringhe.

- a. Diagonale sinistra-destra che usa il valore dell'elemento fornito per costruire una stringa di asterischi in diagonale, da sinistra verso destra e dall'alto verso il basso. Il numero di asterischi deve corrispondere al valore passato alla funzione. Supponendo che il valore sia pari a 8, deve essere restituita la seguente stringa:

```
"*
 *
  *
   *
    *
     *
      *
       *
        *"
```

- b. Diagonale destra-sinistra che usa il valore dell'elemento fornito come parametro per costruire una stringa di asterischi in diagonale, da destra verso sinistra e dall'alto verso il basso. Il numero di asterischi deve corrispondere al valore passato alla funzione. Supponendo che il valore sia pari a 8, deve essere restituita la seguente stringa:

```
"
 *
 *
 *
 *
 *
 *
 *
 *"
```

- c. Triangolo che usa il valore dell'elemento fornito come parametro alla funzione per costruire una stringa di asterischi che formano un triangolo isoscele, con i

due lati uguali composti da un numero di asterischi pari al valore del parametro della funzione. Supponendo che il valore sia pari a 8, deve essere restituita la seguente stringa:

```
"      *
      * *
     *  *
    *   *
   *    *
  *     *
 *      *
*       *
*****"
```

Usare la programmazione modulare per costruire un modulo che contenga tutte le funzioni di creazione dinamica e deallocazione delle stringhe che rappresentano le figure geometriche. La visualizzazione della stringa deve avvenire nella funzione *main()*.

179. Progettare un'applicazione in grado di prendere in input dei valori numerici interi, allocandoli in un array dinamico, e visualizzi delle figure geometriche per ogni elemento dell'array, usando delle funzioni che restituiscono le seguenti stringhe.

- a. Diagonale sinistra-destra che usa il valore dell'elemento corrente per costruire una stringa di asterischi in diagonale, da sinistra verso destra e dall'alto verso il basso. Il numero di asterischi deve corrispondere al valore ricevuto. Supponendo che l'attuale valore dell'array sia pari a 8, deve essere restituita la seguente stringa:

```
" *
  *
 *
 *
 *
 *
 *
 *
  *"
```

- b. Diagonale destra-sinistra che usa il valore dell'elemento corrente per costruire una stringa di asterischi in diagonale, da destra verso sinistra e dall'alto verso il basso. Il numero di asterischi deve corrispondere al valore ricevuto. Supponendo che l'attuale valore dell'array sia pari a 8, deve essere restituita la seguente stringa:

```
"      *
      *
     *
    *
   *
  *
 *
*
*"
```

- c. Triangolo che usa il valore dell'elemento corrente per costruire una stringa di asterischi che formano un triangolo isoscele con i due lati uguali composti da un numero di asterischi pari al valore dell'elemento corrente. Supponendo che l'attuale valore dell'array sia pari a 8, deve essere restituita la seguente stringa:

```
"          *
        * *
       *  *
      *   *
     *    *
    *     *
   *      *
  *       *
 *        *
*****"
```

Usare la programmazione modulare prevedendo quanto segue.

- Un modulo che contenga le funzioni per l'allocazione dinamica (uso della funzione *malloc()*) di un array dinamico di interi e una funzione per la sua deallocazione (uso della funzione *free()*). Se questo modulo è già stato creato per altre applicazioni può essere riutilizzato.
- Un modulo che contenga tutte le funzioni di creazione dinamica delle stringhe che rappresentano le figure geometriche usando gli elementi dell'array per dimensionarle. È importante ricordare di effettuare la deallocazione delle stringhe dinamiche quando non sono più necessarie. La visualizzazione della stringa deve avvenire nella funzione *main()*. Se questo modulo è già stato creato per altre applicazioni può essere riutilizzato.

Progettazione (Liu)

Progettazione strutture dati

Nell'esercizio, vengono utilizzate diverse funzioni per gestire un array di numeri interi inseriti dall'utente e per stampare delle figure geometriche. Le strutture dei dati sono:

1. Array di interi: Viene utilizzato un array di interi per memorizzare i numeri inseriti dall'utente.

Relazioni tra le entità

In questo esercizio non ci sono particolari relazioni tra le entità, perché abbiamo solo un array che non ha rapporto con nessuno.

Casi d'uso

Lo scopo di questo esercizio è chiedere all'utente la dimensione dell'array e, in base alla dimensione che l'utente fornisce, viene richiesto l'inserimento dei numeri. In base ai numeri inseriti dall'utente si procede alla visualizzazione delle diverse figure geometriche.

Creazione array di interi- caso principale

- L'utente inserisce la dimensione
- L'array viene creato dinamicamente usando la dimensione inserita dall'utente

Per svolgere questi operazioni ho bisogno di utilizzare le seguenti funzioni:

- `dimensione(...);`
- `allocaarray(...);`

Creazione array di interi- caso alternativo

- Se la dimensione del array è minore o uguale a zero, viene nuovamente richiesto l'inserimento della dimensione.
- Se l'allocazione del array non avviene correttamente, visualizza errore e si chiude l'applicazione

Caricamento array di interi- caso principale

- Viene effettuato il caricamento dei numeri tramite un ciclo
- Al termine dell'uso dei array deve essere deallocato

Per svolgere questi operazioni ho bisogno di utilizzare le seguenti funzioni:

- `caricamentoarray(...);`
- `deallocaarray(...);`

Caricamento array di interi- caso alternativo

- Se nel caricamento il numero inserito dall'utente è minore o uguale a zero, viene nuovamente richiesto l'inserimento del numero.

Figure geometriche - caso principale

In questo esercizio in base ai numeri inseriti dall'utente si visualizzano le figure geometriche.

- Diagonale sinistra-destra: utilizziamo due cicli for uno dentro l'altro
- Diagonale destra-sinistra: utilizziamo due ciclo for uno dentro l'altro
- Triangolo: visualizziamo il triangolo con * dimensione che l'utente ha scelto.

Per visualizzare questi figure geometriche ho bisogno di utilizzare le seguenti funzioni:

- `Diagonalesinistradestra(...);`
- `Diagonaledestrasinistra(...);`
- `Triangolo(...);`

Figure geometriche - casi alternati

In questo caso d'uso non ci sono caso alternativi, perché erano tutte funzioni di output.

180. Usando il codice presentato dal docente sull'[allocazione dinamica di un array di stringhe](#), aggiungere una funzione di ordinamento che permetta di ordinare i nomi in ordine lessicografico crescente. Per confrontare due stringhe per individuare il loro ordine lessicografico devi usare la funzione [strcmp\(\)](#).

Funzione da aggiungere nel file [stringhe.c](#) (Bolognone)

```
void ordinaArrayStringhe(char **array, int numStringhe) {
    for(int i=0; i<numStringhe-1; i++){
        for(int j=0; j<numStringhe-i-1; j++){
            if(strcmp(array[j], array[j+1]) > 0){
                char *temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }
}
```

```
    }  
  }  
}
```

181. Usando il codice dell'esercizio 169, aggiungere una funzione che permetta di ridimensionare a piacimento l'array di stringhe. Per ridimensionare un array dinamico devi usare la funzione `realloc()`, spiegata anche dal video nel punto (e) del paragrafo [Array dinamico](#).
182. Progettare una funzione che, a partire da un array numerico di dimensione N fornito come parametro, copi in un secondo array (anch'esso fornito come parametro) i soli elementi del primo array compresi tra un valore minimo a e un valore massimo b (anch'essi passati come argomenti). La funzione deve restituire la dimensione del secondo array.
Progettare quindi un algoritmo che, utilizzando la funzione precedente, visualizzi gli elementi del secondo array dopo avere richiesto l'inserimento da tastiera dei valori del primo array.
183. Progettare e implementare una libreria per la rappresentazione dei numeri in base 10 nella codifica IEEE-754 a 32 bit.

La libreria deve permettere di ottenere la codifica IEEE-754 del numero sia in base 2, sia in base 16.

La libreria deve prevedere una struttura dati adeguata per rappresentare ogni singola parte del numero finale.

Le funzioni necessarie per la conversione possono essere le seguenti:

- una funzione che determini il segno del numero;
- una funzione che converta in base 2 un numero in base 10 sia per la parte intera, sia per la parte frazionaria;
- una funzione che normalizzi la mantissa con bit hidden e che restituisca il valore dell'esponente;
- una funzione che calcoli l'esponente eccesso 127 in binario;
- una funzione che restituisce la rappresentazione binaria della codifica IEEE-754 del numero in base 10 fornito;
- una funzione che restituisce la rappresentazione esadecimale della codifica IEEE-754 del numero in base 10 fornito.

Le uniche funzioni pubbliche della libreria devono essere quelle del punto (e) e del punto (f). Tutte le altre funzioni devono essere private alla libreria.

184. I codici a barre dei prodotti sono composti da 13 cifre di cui l'ultima è la cifra di controllo che si determina a partire dalle prime 12 (il codice vero e proprio) con le seguenti regole:
- moltiplicare per 3 tutte le cifre in posizione pari (la seconda, la quarta, ... fino alla dodicesima);
 - moltiplicare per 1 tutte le cifre in posizione dispari (la prima, la terza, ... fino alla undicesima) (questa operazione non cambia i valori nelle posizioni dispari);
 - sommare i 12 valori così ottenuti;

- d. prendere il resto della divisione per 10 della somma ottenuta;
- e. sottrarre da 10 il resto della divisione precedente; se quest'ultimo valore fosse pari a 10, lo si sostituisca con 0.

Scrivere una funzione che, a partire da un vettore di 12 numeri corrispondenti alle singole cifre di un codice a barre, restituisca la cifra di controllo calcolata con le regole illustrate.

Progettare quindi un algoritmo che richieda all'utente l'inserimento delle singole cifre di un codice a barre e visualizzi la corrispondente cifra di controllo calcolata con la funzione realizzata in precedenza.

185. Progettare un algoritmo che, fornendo in input le prime 12 cifre di un codice a barre, svolga le seguenti operazioni:
- a. calcoli la cifra di controllo e la aggiunga al codice a barre iniziale (vedere l'esercizio 77);
 - b. visualizzi la scritta "Libro" se il codice a barre è un codice ISBN o la scritta "Altro" se il codice non identifica un libro e concludere l'algoritmo; per effettuare questo controllo dovrete controllare il [prefisso EAN](#);
 - c. visualizzi la stringa del [gruppo linguistico](#) supponendo di considerare solo i codici composti da due cifre da 80 a 94 (utilizzare un array per gestire il gruppo linguistico);
 - d. visualizzi il nome della [casa editrice](#) supponendo di considerare solo i codici da 00 a 19 (utilizzare un array per gestire le case editrici).

Si usino opportunamente le funzioni per svolgere l'esercizio (caricamento array, calcolo della cifra di controllo, gestione prefisso EAN, individuazione del gruppo linguistico, individuazione della casa editrice).

186. In un array di 9 elementi sono memorizzate le presenze mensili degli studenti di una classe durante l'anno scolastico. Progettare un algoritmo che determini:
- a. la media mensile di presenze nel corso dell'intero anno scolastico;
 - b. il numero totale di presenze nel primo quadrimestre;
 - c. il mese in cui si è registrato il numero massimo di presenze;
 - d. il mese in cui si è registrato il numero minimo;
 - e. la media delle presenze del secondo quadrimestre.
187. Progettare l'algoritmo di ordinamento Selection Sort e verificarne il funzionamento utilizzando un array di N elementi fornito in input. Si utilizzino le funzioni per gli input, per l'algoritmo Selection Sort e per l'output.
188. Progettare l'algoritmo di ordinamento Bubble Sort e verificarne il funzionamento utilizzando un array di N elementi fornito in input. Si utilizzino le funzioni per gli input, per l'algoritmo Bubble Sort e per l'output.
189. Progettare due funzioni distinte che permettano di ordinare in ordine crescente un array di interi e un array di stringhe usando l'algoritmo Bubble Sort in entrambi i casi. Per testare le due funzioni creare un progetto dove nella funzione main() si effettuerà il caricamento disordinato dei due array.
190. (*Chiesta progettazione*) Si vogliono organizzare i dati relativi ai risultati di una corsa campestre sulla distanza dei 1000 metri. Gli iscritti sono meno di 10(0) e si intende memorizzare il numero di gara, il nominativo e il tempo in secondi impiegato per portare a termine la prova. I dati dovranno essere memorizzati in un singolo array

definendo un nuovo tipo per gestire i diversi iscritti alla corsa. L'algoritmo deve prevedere un menù che permetta di svolgere le seguenti operazioni:

- a. caricamento dei dati degli atleti;
- b. visualizzazione della classifica ottenuta ordinando i dati degli atleti per creare la classifica finale in base ai tempi di arrivo (*selection sort* o *bubble sort*);
- c. uscita dal programma.

Si usi la programmazione modulare per gestire tutti i distinti aspetti logici dell'algoritmo in un'ottica di progettazione top-down. Le strutture dati devono essere tutte gestite dinamicamente.

Progettazione Strutture Dati (Lipari)

Le strutture dati utilizzate saranno:

- Una struttura Competitore per memorizzare le informazioni di ciascun partecipante (numero di gara, nome, tempo).
- Un array dinamico di strutture Competitore per memorizzare tutti i partecipanti alla corsa.

```
struct Competitore {  
    int numero_gara;  
    char nome[LEN_NOME];  
    char cognome[LEN_COGNOME];  
    int tempo;  
};  
  
struct Competitore* competitori;
```

Relazioni tra le Entità

Non esiste una relazione con altra entità, viene gestito solo l'array dinamico di Competitore che aggrega un insieme di partecipanti.

Casi d'uso

Menù - Caso d'uso principale

- Il sistema visualizza il menù delle operazioni disponibili.
- L'utente inserisce una scelta
- Svolta l'operazione scelta dall'utente, il menù viene nuovamente visualizzato, tranne nel caso in cui non sia stata scelta l'uscita dall'applicazione. In quest'ultimo caso l'applicazione termina.

```
1) Carica partecipanti alla gara  
2) Visualizza classifica  
3) Uscita
```

Scelta: _

Menù - Caso d'uso alternativi

- Se l'utente inserisce un'opzione non valida, il sistema visualizza un messaggio di errore e visualizza nuovamente il menù per richiedere una nuova scelta.

- Se l'utente chiede di visualizzare la classifica prima di aver effettuato il caricamento dei partecipanti, viene visualizzato un messaggio di errore e riproposto il menù.

Caricamento dei dati dei Partecipanti - Caso d'uso principale

- Il sistema chiede qual è il numero dei partecipanti.
- Il sistema chiede all'utente di inserire i dati di ciascun partecipante (numero di gara, nome, cognome, tempo).
- I dati inseriti vengono memorizzati nell'array dei competitori.

Inserimento numero partecipanti.

Numero partecipanti: 4

Inserimento dati partecipante.

Numero di gara: 1
Nome: Pippo
Cognome: Pippoli
Tempo di gara: 112

Caricamento dei Dati dei Partecipanti - Casi alternativi

Numero partecipanti - Casi alternativi

- Se il numero di partecipanti è negativo o pari a zero, oppure è superiore o pari a 100, viene visualizzato un messaggio di errore e viene richiesto l'inserimento.

Numero partecipanti: -2

Errore. Il numero dei partecipanti deve essere un valore numerico compreso nell'intervallo [1, 99]

- Se per il numero di partecipanti venisse inserito un carattere, viene segnalato l'errore e richiesto nuovamente l'inserimento.

Numero partecipanti: k

Errore. Il numero dei partecipanti deve essere un valore numerico compreso nell'intervallo [1, 99]

Numero di gara partecipante - Casi alternativi

- Se il numero di gara del partecipante è negativo o pari a zero, oppure è superiore al numero dei partecipanti, viene visualizzato un messaggio di errore e viene richiesto l'inserimento.

Numero di gara: 8

Errore. Il numero di gara deve essere un valore numerico compreso nell'intervallo [1, 4]

Numero di gara: _

- Se per il numero di gara del partecipante venisse inserito un carattere, viene segnalato l'errore e richiesto nuovamente l'inserimento.

Numero di gara: 8

Errore. Il numero di gara deve essere un valore numerico compreso nell'intervallo [1, 4]

Numero di gara: _

- Se il numero di gara è un valore che è già stato inserito, viene segnalato l'errore e richiesto nuovamente l'inserimento.

Numero di gara: 1

Errore. Il numero di gara è già stato inserito

Numero di gara: _

Tempo di gara del partecipante - Casi alternativi

- Se il tempo di gara del partecipante è negativo o pari a zero, viene visualizzato un messaggio di errore e viene richiesto l'inserimento.

Numero di gara: 1

Nome: Pippo

Cognome: Pippoli

Tempo di gara: -34

Errore. Il tempo di gara deve essere un valore numerico positivo

Tempo di gara: _

- Se per il tempo di gara del partecipante venisse inserito un carattere, viene segnalato l'errore e richiesto l'inserimento.

Numero di gara: 1

Nome: Pippo

Cognome: Pippoli

Tempo di gara: d

Errore. Il tempo di gara deve essere un valore numerico positivo

Tempo di gara: _

Visualizzazione della Classifica - Casi principali

- Il sistema crea la classifica finale ordinando i partecipanti in base ai tempi di arrivo.
- Successivamente, mostra i dati dei partecipanti in formato tabellare nell'ordine stabilito dall'ordinamento.

N.gara	Nome	Cognome	Tempo
1	Pippo	Pippoli	112
4	Minni	Minnoli	113
2	Pluto	Plutoli	120
3	Paolino	Paperino	135

191. (*Chiesto codice*) Usando il codice presentato dal docente sull'[allocazione dinamica di un array di stringhe](#) con l'aggiunta della funzione di ordinamento, modificare l'applicazione aggiungendo un array dinamico di float che contenga i pesi corrispondenti di ogni singola persona caricata nell'array di stringhe.

I due array paralleli devono essere gestiti insieme, quindi si aggiungano due ulteriori funzioni di ordinamento:

- una funzione di ordinamento crescente dei nomi che gestisca anche lo spostamento del corrispondente peso della persona;
- una funzione di ordinamento decrescente dei pesi che gestisca anche lo spostamento del corrispondente nome della persona.

Per confrontare due stringhe in modo da individuare il loro ordine lessicografico si usi la funzione [strcmp\(\)](#).

Si usi la programmazione modulare per gestire tutti i distinti aspetti logici dell'algoritmo in un'ottica di progettazione top-down. Le strutture dati devono essere tutte gestite dinamicamente.

192. (*Chiesta progettazione*) Progettare un'applicazione che permetta di gestire i libri di un negozio.

Di ogni libro si devono memorizzare il titolo, il prezzo, l'anno di pubblicazione e gli autori.

Per gestire al meglio tutti i dati il negozio memorizza separatamente l'elenco degli autori e un libro può avere un numero arbitrario di autori.

Per ogni autore prevede la memorizzazione del nome, del cognome, del luogo e della data di nascita.

L'applicazione deve permettere di caricare i dati degli autori e, successivamente si può effettuare il caricamento dei libri.

L'applicazione deve permettere di effettuare le seguenti operazioni:

- visualizzazione in ordine alfabetico dell'elenco dei libri;
- visualizzazione in ordine alfabetico degli autori;
- dato il titolo di un libro visualizzazione di tutti i suoi dati;
- dato il nome di un autore visualizzazione di tutti i libri che ha scritto;

Si usi la programmazione modulare per gestire tutti i distinti aspetti logici dell'algoritmo in un'ottica di progettazione top-down. Le strutture dati devono essere tutte gestite dinamicamente.

193. Progettare un'applicazione che permetta di gestire i libri di un negozio

Di ogni libro si devono memorizzare il titolo, il prezzo, l'anno di pubblicazione, gli autori, l'editore, i generi di appartenenza del libro e il codice ISBN.

Per gestire al meglio tutti i dati il negozio memorizza separatamente l'elenco degli autori, degli editori e dei generi.

Per ogni autore prevede la memorizzazione del nome, del cognome, del luogo e della data di nascita.

Per gli editori memorizza la ragione sociale e l'indirizzo della sede centrale.

Per i generi si memorizza solo la descrizione del genere.

Un libro può avere un numero arbitrario di autori e di generi, ma l'editore è unico.

L'applicazione deve permettere di caricare i dati degli autori, degli editori e dei generi e, successivamente si può effettuare il caricamento dei libri.

L'applicazione deve permettere di effettuare le seguenti operazioni:

- a. visualizzare in ordine alfabetico l'elenco dei libri;
- b. visualizzazione in ordine alfabetico degli autori;
- c. visualizzazione in ordine alfabetico degli editori;
- d. dato il titolo di un libro visualizzazione di tutti i suoi dati;
- e. dato il nome di un autore visualizzazione di tutti i libri che ha scritto;
- f. data la ragione sociale di un editore visualizzazione di tutti i libri pubblicati;
- g. dato un genere, visualizzazione di tutti i libri che appartengono al genere indicato.

Si usi la programmazione modulare per gestire tutti i distinti aspetti logici dell'algoritmo in un'ottica di progettazione top-down. Le strutture dati devono essere tutte gestite dinamicamente.

194. Progettare un algoritmo che permetta di gestire una rubrica telefonica. L'algoritmo dovrà prevedere quanto di seguito elencato usando le funzioni:

- a. la gestione tramite menù degli aspetti indicati di seguito;
- b. caricamento di un nominativo nella rubrica in modo ordinato (*insertion sort* o riordino tramite *bubble sort* ad ogni inserimento);
- c. visualizzazione dell'intera rubrica;
- d. ricerca del numero di telefono di un contatto attraverso il nominativo (*ricerca dicotomia/binaria*);
- e. cancellazione di un contatto dalla rubrica (devono essere eliminati sia il nominativo, sia il numero telefonico).

Si usi la programmazione modulare per gestire tutti i distinti aspetti logici dell'algoritmo in un'ottica di progettazione top-down. Le strutture dati devono essere tutte gestite dinamicamente.

195. Progettare un algoritmo che permetta di gestire le squadre e gli allenatori del Fantacalcio. L'algoritmo deve permettere di:

- a. inserire i dati delle squadre e dei relativi allenatori;
- b. visualizzare il cognome degli allenatori delle sole squadre che hanno totalizzato più di 30 punti, in ordine di classifica, cioè partendo dalla squadra che ha totalizzato più punti;
- c. il numero totale di allenatori che hanno vinto almeno un titolo.

Una squadra di calcio è identificata dalle seguenti informazioni:

- e. nome della squadra;
- f. colore della maglia;

- g. punteggio corrente;
- h. allenatore.

L'allenatore è una persona caratterizzata da:

- c. nome e cognome;
- d. numero di titoli vinti.

Matrici e funzioni

196. Implementa il gioco della battaglia navale semplificato. Il programma deve generare in autonomia il campo di gioco ponendo in modo pseudo casuale 10 navi che occupano una sola cella, facendo attenzione che non siano adiacenti. Il giocatore deve introdurre le coordinate del campo di gioco per indicare il suo tentativo e deve cercare di affondare le navi in 20 tentativi (o altro numero a tua scelta). Il campo di gioco deve essere disegnato in modo testuale. Il gioco deve prevedere un singolo giocatore che gioca con un campo preparato dal computer.

Per svolgere l'esercizio sviluppa poco per volta le diverse funzionalità tramite dei moduli e delle funzioni apposite, considerando le seguenti caratteristiche.

- a. Individuazione delle strutture dati necessarie per il gioco. La battaglia navale prevede l'uso di una matrice 10x10 e 10 navi disposte in questo spazio.
- b. Scelta dei simboli per rappresentare lo stato di una cella visibile al giocatore. Puoi usare il simbolo '~' per mostrare il campo di gioco iniziale, il simbolo '*' per presenza di nave colpita e affondata e il simbolo 'o' per un tentativo andato a vuoto. Di seguito viene mostrato un campo di gioco visibile al giocatore dove sono state colpite e affondate 4 navi su 10, mentre 5 tentativi sono andati a vuoto.

```

  1 2 3 4 5 6 7 8 9 10
A ~ ~ ~ * ~ ~ ~ ~ ~
B ~ ~ ~ ~ ~ ~ * ~ ~
C ~ o ~ ~ ~ ~ ~ ~ o ~
D ~ ~ ~ ~ ~ ~ ~ ~ ~
E ~ ~ ~ ~ ~ ~ ~ ~ ~
F ~ ~ ~ ~ ~ ~ ~ ~ ~
G ~ ~ * o ~ ~ ~ ~ ~
H ~ ~ ~ ~ ~ ~ ~ ~ ~
I ~ ~ ~ ~ ~ ~ ~ * ~
J o ~ ~ ~ ~ ~ ~ ~ ~ o

```

Inserisci le coordinate: _

- c. Il computer deve posizionare le navi sul campo di gioco non visibile all'utente e si può usare il simbolo '^' per presenza di nave non colpita. In alternativa, se non vuoi usare una matrice per il campo di gioco non visibile, puoi memorizzare le coordinate come posizione, ad esempio B7, usando un semplice array.
- d. Posizionamento delle navi nel tavolo di gioco. Il computer deve scegliere in modo casuale il posizionamento delle navi verificando che non siano adiacenti l'una all'altra, cioè deve esserci almeno un quadrato vuoto del campo di gioco che distanzia le navi. Per semplificare questa operazione può essere considerato un campo di gioco che abbia una cornice di celle aggiuntive che non vengono mai usate per posizionare le navi. Questo rende il controllo del distanziamento delle navi più semplice. Se trovi una soluzione diversa usala a tua discrezione.
- e. Visualizzazione del campo di gioco visibile al giocatore. Il campo di gioco visibile al giocatore nasconde ovviamente la posizione delle navi e rende visibili solo quelle affondate e i tentativi non andati a buon fine. La visualizzazione del campo di gioco del giocatore deve essere aggiornata ogni volta che vengono inserite delle coordinate per tentare di colpire e affondare una nave. Ad esempio, supponendo che nelle coordinate fornite di seguito ci sia una nave la visualizzazione deve cambiare nel seguente modo.

```

  1 2 3 4 5 6 7 8 9 10
A ~ ~ ~ * ~ ~ ~ ~ ~
B ~ ~ ~ ~ ~ ~ * ~ ~
C ~ o ~ ~ ~ ~ ~ o ~
D ~ ~ ~ ~ ~ ~ ~ ~ ~
E ~ ~ ~ ~ ~ ~ ~ ~ ~
F ~ ~ ~ ~ ~ ~ ~ ~ ~
G ~ ~ * o ~ ~ ~ ~ ~
H ~ ~ ~ ~ ~ ~ ~ ~ ~
I ~ ~ ~ ~ ~ ~ ~ * ~
J o ~ ~ ~ ~ ~ ~ ~ o

```

Inserisci le coordinate: **E6**

```

  1 2 3 4 5 6 7 8 9 10
A ~ ~ ~ * ~ ~ ~ ~ ~
B ~ ~ ~ ~ ~ ~ * ~ ~
C ~ o ~ ~ ~ ~ ~ o ~
D ~ ~ ~ ~ ~ ~ ~ ~ ~
E ~ ~ ~ ~ ~ * ~ ~ ~
F ~ ~ ~ ~ ~ ~ ~ ~ ~
G ~ ~ * o ~ ~ ~ ~ ~
H ~ ~ ~ ~ ~ ~ ~ ~ ~
I ~ ~ ~ ~ ~ ~ ~ * ~
J o ~ ~ ~ ~ ~ ~ ~ o

```

Inserisci le coordinate: _

Nel caso in cui invece non ci fosse una nave, la visualizzazione deve cambiare nel seguente modo.

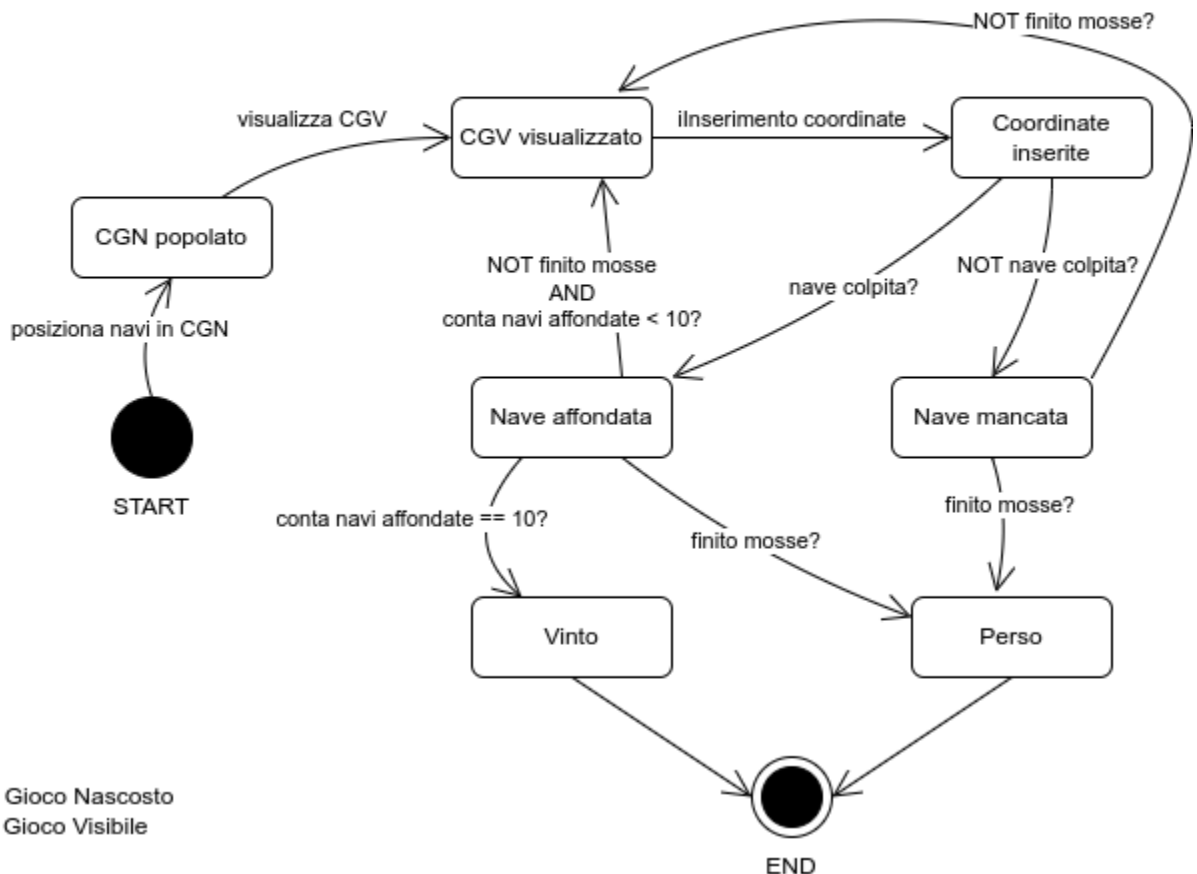
	1	2	3	4	5	6	7	8	9	10
A	~	~	~	*	~	~	~	~	~	~
B	~	~	~	~	~	~	*	~	~	~
C	~	o	~	~	~	~	~	~	o	~
D	~	~	~	~	~	~	~	~	~	~
E	~	~	~	~	~	o	~	~	~	~
F	~	~	~	~	~	~	~	~	~	~
G	~	~	*	o	~	~	~	~	~	~
H	~	~	~	~	~	~	~	~	~	~
I	~	~	~	~	~	~	*	~	~	~
J	o	~	~	~	~	~	~	~	~	o

Inserisci le coordinate: _

Di seguito viene fornito l'automa a stati finiti che descrive il gioco della battaglia navale.

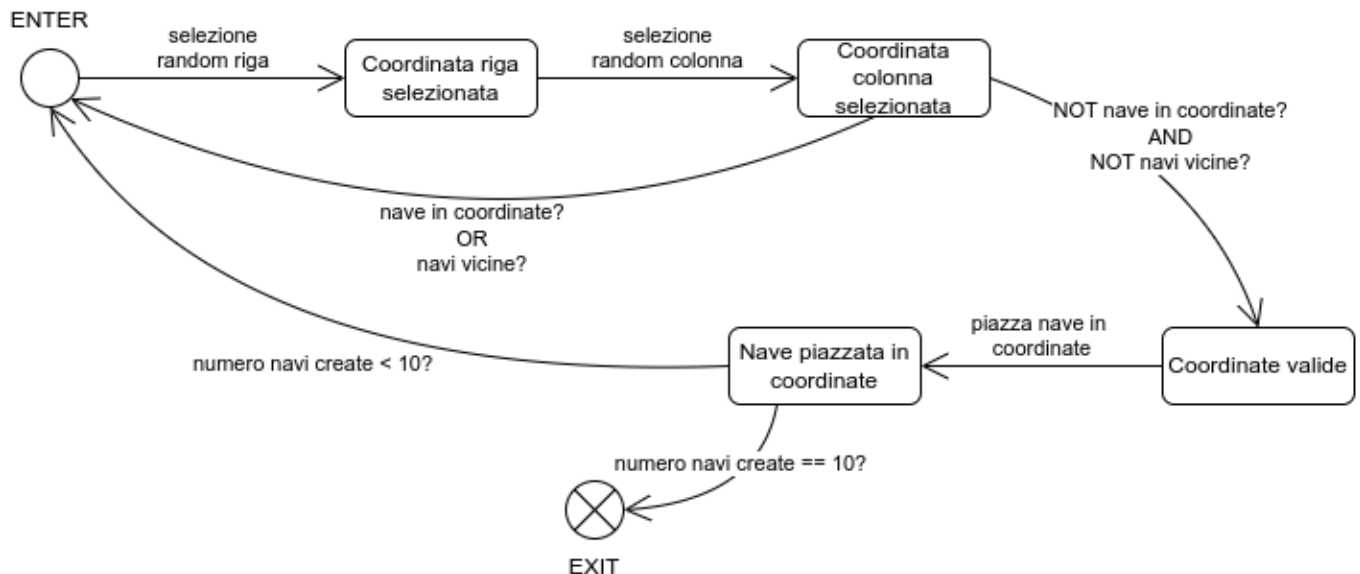
Battaglia navale

Macchina a stati finiti che descrive il gioco della battaglia navale.



Posizionamento navi in CGN

Macchina a stati finiti per il posizionamento delle navi nel CGN



197. Implementare il gioco della battaglia navale. Il programma deve generare in autonomia il tavolo di gioco ponendo in modo pseudo casuale tutte le navi (una nave di 4 posizioni, 2 di 3 posizioni, 3 di due posizioni, 4 di una posizione). Le navi possono essere disposte verticalmente o orizzontalmente sul tavolo di gioco e non possono essere adiacenti. Il giocatore deve introdurre le coordinate del tavolo di gioco per indicare il suo tentativo. Si disegni in modo testuale il tavolo di gioco. Il gioco deve prevedere un singolo giocatore che gioca con un tavolo preparato dal computer. Per lo sviluppo delle diverse fasi si faccia riferimento all'esercizio precedente.

198. Relativamente al progetto dell'IMC già visto in precedenza, si progetti un algoritmo che usi una matrice per memorizzare il numero di persone che risultano sottopeso (s), normopeso (n), sovrappeso (S) e obeso (o), utilizzando il carattere '*' per ogni persona che rientri in uno dei gruppi elencati. Si visualizzi infine l'istogramma orizzontale usando la matrice usata per memorizzare i dati. Esempio: sottopeso = 3, normopeso = 5, sovrappeso = 8, obeso = 2

```

s:  ***
n:  *****
S:  *****
o:  **
  
```

199. Scrivere una funzione che determini, producendo come risultato true o false, se una matrice quadrata di $N \times N$ elementi è simmetrica, cioè se gli elementi in posizione simmetrica rispetto alla diagonale principale sono tutti uguali.

200. Scrivere una funzione che costruisca, in una matrice quadrata di $N \times N$ elementi passata come argomento, la trasposta di una matrice quadrata di $N \times N$ elementi fornita anch'essa come argomento. La trasposta di una matrice è una matrice in cui le posizioni degli elementi sono modificate scambiando gli indici di riga e di colonna, per

$$\begin{pmatrix} 2 & 1 & 5 & 9 \\ 0 & 1 & 8 & 4 \\ 2 & 1 & 9 & 3 \\ 6 & 7 & 0 & 2 \end{pmatrix}$$

esempio la trasposta della seguente matrice
è la matrice

$$\begin{pmatrix} 2 & 0 & 2 & 6 \\ 1 & 1 & 1 & 7 \\ 5 & 8 & 9 & 0 \\ 9 & 4 & 3 & 2 \end{pmatrix}$$

Scrivere un algoritmo che, utilizzando la funzione precedente e costruendo la matrice iniziale a partire da valori forniti da tastiera, visualizzi la matrice trasposta disponendo gli elementi su più righe.

201. bla bla

Funzioni, ricorsione, array e matrici

202. Scrivere una funzione ricorsiva che calcoli la somma degli elementi di un array di n interi.

203. Scrivere un algoritmo ricorsivo per il calcolo del minimo in un array di n interi positivi.

204. Scrivere una funzione ricorsiva che, avendo in input un array di n interi, dia in output TRUE se 10 è un elemento della lista, FALSE altrimenti.

205. Scrivere una funzione ricorsiva che, avendo in input un array di n interi, dia in output TRUE se tutti gli elementi sono maggiori di 10, FALSE altrimenti.

206. Scrivere una funzione ricorsiva $\text{pot}(n)$ per il calcolo dei numeri $f(n)$ definiti dalle seguenti relazioni:

$$f(1) = 2$$

$$f(n) = 2 * f(n - 1) \text{ per } n \geq 2$$

207. Scrivere una funzione ricorsiva $\text{time}(n)$ per il calcolo dei numeri $t(n)$ definiti dalle seguenti relazioni:

$$t(0) = 0$$

$$t(1) = 1$$

$$t(n) = 2 * t(n - 2) + 5 \text{ per } n \geq 2$$

208. Scrivere una funzione ricorsiva che, avendo in input un array di n interi, dia in output il numero degli elementi positivi della lista.

209. Scrivere una funzione ricorsiva $\text{time}(n)$ per il calcolo dei numeri $t(n)$ definiti dalle seguenti relazioni:

$$t(0) = 0$$

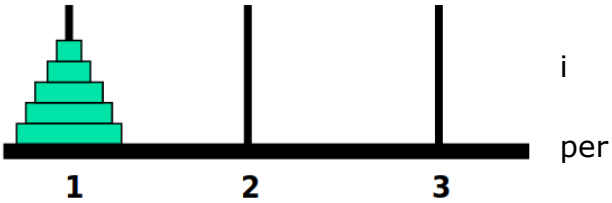
$$t(1) = 1$$

$$t(n) = 2 * t(n - 2) + t(n - 1) \text{ per } n \geq 2$$

210. [Ricorsione non lineare (la definizione della funzione si basa su più chiamate ricorsive)] Progettare un algoritmo per il calcolo del MCD fra due numeri utilizzando la seguente specifica ricorsiva:

$$\begin{aligned} \text{MCD}(m,n) &= n && \text{se } n = m \\ \text{MCD}(m,n) &= \text{MCD}(m-n, n) && \text{se } m > n \\ \text{MCD}(m,n) &= \text{MCD}(m, n-m) && \text{se } m < n \end{aligned}$$

211. Scrivere una funzione ricorsiva `writeVertical(int n)` che prende un intero e lo stampa a video con tutte le cifre incolonnate, a partire da quella più significativa.
212. Scrivere una funzione ricorsiva `writeVertical(int n)` che prende un intero e lo stampa a video con tutte le cifre incolonnate, a partire da quella meno significativa.
213. Scrivere un programma ricorsivo che prenda un numero intero n e stampi un numero di asterischi uguale alla somma dei quadrati dei primi n numeri naturali.
214. Scrivere un programma ricorsivo che calcoli la seguente funzione:
$$\begin{aligned} f(n) &= n * n + f(n-2) \\ f(1) &= 1 \\ f(0) &= 0 \end{aligned}$$
215. Considerando che le operazioni aritmetiche di somma e prodotto sui numeri naturali sono definite in modo ricorsivo:
$$\begin{aligned} n + 0 &= n \\ n + m &= \text{successore}(n + (m-1)) \\ n * 0 &= 0 \\ n * m &= (n * (m-1)) + n \end{aligned}$$
si scriva un programma ricorsivo che le svolga in modo ricorsivo.
216. Spiegare cosa succede con la seguente funzione ricorsiva, facendone un esempio:
$$\begin{aligned} f(n) &= f(n+1) + 1 \\ f(0) &= 0 \end{aligned}$$
217. Scrivere una funzione ricorsiva che prenda in input una sequenza di caratteri che termina con il "." e la stampa in maniera invertita. La lunghezza della sequenza è indefinita.
218. Scrivere un programma ricorsivo che prenda in input una sequenza di cifre positive con al centro il valore 0, e dica se la sequenza è palindroma o meno. Si assuma che le lunghezze delle sequenze prima dello 0 e dopo lo 0 siano uguali.
219. Scrivere un programma ricorsivo che prenda in input una sequenza di cifre positive con al centro il valore 0, e dica se la sequenza è palindroma o meno. Si assuma che la lunghezza delle sequenze prima e dopo lo 0 possa essere diversa e che la sequenza termini con un ".".
220. Scrivere una funzione ricorsiva che prende una lista di interi e ritorni la somma degli interi nella lista.
221. Scrivere una funzione ricorsiva che prende una lista e ritorna la lista invertita.
222. Scrivere una funzione ricorsiva che prende una lista ed un intero e rimuove dalla lista la prima occorrenza dell'intero.
223. Scrivere una funzione ricorsiva che prende una lista ed un intero e rimuove dalla lista tutti i multipli dell'intero.

224. Scrivere¹⁴ una funzione ricorsiva per determinare se una stringa è palindroma.
225. Scrivere una funzione ricorsiva per visualizzare una stringa a rovescio senza che la stringa sia effettivamente invertita.
226. Scrivere un programma per risolvere il problema delle torri di Hanoi: spostare tutti dischi da 1 a 2 usando 3 come temporaneo. Si può spostare un solo pezzo volta. Un pezzo grande non può stare sopra uno più piccolo.
- 
- Suggerimento:* usare la funzione `muovi(quantità, from, to, temp)` che muove un disco direttamente da `from` a `to` solo se `quantità` vale 1.
227. Scrivere una funzione ricorsiva per cercare un valore all'interno di un vettore non ordinato (ricerca lineare). Risultato: -1 se non lo trova, altrimenti l'indice della posizione dove è stato trovato.
228. Scrivere una funzione ricorsiva per cercare un valore all'interno di un vettore ordinato (ricerca dicotomica). Risultato: -1 se non lo trova, altrimenti l'indice della posizione dove è stato trovato.
229. Scrivere una funzione che realizzi un selection sort (ordinamento crescente di un array) in modo ricorsivo su un vettore di interi.
230. Si scriva una funzione C ricorsiva con un parametro `n` di tipo intero che calcoli l'`n`-esimo numero della serie di Fibonacci. Si scriva nel commento introduttivo al codice, cosa calcola la funzione nel caso che il numero sia negativo.
231. bla bla

¹⁴ [Claudio Fornaro, La ricorsione \(introduzione\)](#), ultimo accesso novembre 2022

Problemi vari

232. Progettare un algoritmo che permetta di rappresentare le [quote ortometriche](#) (geoidiche) di un territorio rettangolare utilizzando una matrice, e di derivare le seguenti informazioni:

- a. la quota minima;
- b. la quota massima;
- c. la quota media;
- d. la percentuale della superficie del territorio che ha una quota superiore alla media.

L'algoritmo dovrà permettere di caricare i valori delle quote ortometriche nella matrice e di visualizzare i valori dei punti sopra indicati.

Si usino le funzioni per gestire tutti i distinti aspetti logici dell'algoritmo in un'ottica di progettazione top-down.

233. Il grafo della rete della metropolitana di Parigi, costituita da 298 stazioni, è rappresentato mediante una matrice quadrata di 298×298 elementi contenenti la distanza in chilometri tra le stazioni, che sarà diversa da zero solo se le stazioni risultano direttamente connesse. Per esempio, l'elemento di coordinate [99, 101] conterrà la distanza in chilometri tra la stazione numero 99 e la stazione numero 101 se queste sono direttamente connesse, altrimenti l'elemento conterrà il valore zero. Si osservi che l'elemento [99, 101] dovrà necessariamente essere uguale all'elemento [101, 99] (matrice simmetrica).

Progettare i seguenti algoritmi relativi alla metropolitana di Parigi:

- a. caricamento su file dei dati della matrice che rappresenta la rete metropolitana di Parigi;
- b. calcolo e visualizzazione della lunghezza dell'intera rete della metropolitana;
- c. dato il numero di una stazione si elenchino tutte le stazioni che possono raggiungerla.

Si gestiscano le diverse scelte dell'applicazione tramite un menù.

234. La codifica crittografica di Cesare consiste nel sostituire ogni carattere con quello che si trova nella k-esima posizione successiva dell'alfabeto. Il numero k rappresenta la chiave crittografica ($0 < k < 26$). Ad esempio, se $k = 2$ la sostituzione da effettuare sarà 'a' $\rightarrow$ 'c', 'e' $\rightarrow$ 'g', 'z' $\rightarrow$ 'b'. I simboli delle cifre numeriche e i segni di punteggiatura rimangono invece invariati.

Progettare un algoritmo che permetta di:

- a. codificare una stringa secondo la codifica di Cesare in base alla chiave k fornita in input, visualizzando il risultato;
- b. decodificare una stringa codificata in precedenza con la regola di Cesare in base ad una chiave k, visualizzando il testo originale.

Si usi un menù per scegliere l'operazione da fare.

235. La mappa di un territorio viene schematizzata mediante una matrice di dimensioni $M \times N$ all'interno della quale ogni elemento contiene un numero intero che rappresenta in metri l'altezza del terreno in quella zona rispetto al livello del mare (quota ortometrica o geoidica). In un elemento di cui sono note le coordinate di riga e di colonna è presente una sorgente d'acqua. Sapendo che l'acqua può scendere in tutte le direzioni (verticale, orizzontale, diagonale) verso punti con quota minore o uguale, ma non può risalire verso punti di quota superiore, fornire la mappa delle zone che

verranno allagate, per esempio rappresentando gli elementi allagati con un asterisco e quelli non allagati con un punto.

236. Un'immagine digitale a colori è una matrice di pixel, ognuno dei quali è caratterizzato dall'intensità percentuale dei tre colori primari (rosso, verde e blu). La funzione di conversione dal colore al corrispondente livello di grigio tipico delle immagini in bianco e nero è data dalla seguente formula:

$$\text{gray} = (6 * \text{green} + 3 * \text{red} + 1 * \text{blue}) / 10$$

dove green, red e blue sono le intensità dei tre colori primari del pixel preso in considerazione.

Si definisca opportunamente la struttura dati da utilizzare per la rappresentazione di un'immagine a colori e, successivamente si progetti un algoritmo che converta un'immagine a colori in bianco e nero.

237. Progettare un algoritmo per il calcolo del MCD fra due numeri utilizzando la seguente specifica ricorsiva:

$$\text{MCD}(m, n) = n \quad \text{se } n = m$$

$$\text{MCD}(m, n) = \text{MCD}(m - n, n) \quad \text{se } m > n$$

$$\text{MCD}(m, n) = \text{MCD}(m, n - m) \quad \text{se } m < n$$

238. Progettare un algoritmo che permetta di caricare e salvare su file i dati dei corsi universitari tenuti da diversi docenti, con il relativo voto medio degli esami. L'algoritmo deve permettere di

- a. visualizzare l'elenco dei corsi con il nome del docente e il voto medio degli esami in ordine crescente di voto medio;
- b. visualizzare il corso e il voto medio di un docente.

Si facciano tutte le ipotesi aggiuntive sulle strutture dati da utilizzare e si scelga come algoritmo di ordinamento e di ricerca uno fra quelli studiati a lezione. Come algoritmo di ricerca si usi l'algoritmo di ricerca binario.

Si usino le funzioni per gestire tutti i distinti aspetti logici dell'algoritmo in un'ottica di progettazione top-down.

239. Una nota azienda di vendita online ti chiede un software per gestire le vendite online dei suoi prodotti. In particolare il software deve permettere:

- a. all'azienda di gestire l'elenco dei suoi prodotti;
- b. ai clienti di registrarsi al sito di vendita;
- c. agli acquirenti di gestire il proprio carrello.

Per il caricamento dei prodotti indicato nel punto (a), l'applicazione deve prevedere per ogni prodotto un codice, una descrizione e un prezzo unitario. Se reputi necessari altri campi fornisci un'adeguata giustificazione tramite delle ipotesi aggiuntive.

Per la registrazione dei clienti al sito prevista dal punto (b), l'applicazione deve prevedere l'inserimento dei dati anagrafici necessari per identificare il cliente, inviargli la merce acquistata, contattarlo per informarlo delle spedizioni o per segnalare eventuali anomalie, permettergli di recuperare le credenziali di accesso, autenticarsi con più fattori. Inoltre ogni cliente deve poter scegliere più metodi di pagamento tra quelli resi disponibili dall'azienda. Ipotizza giustificando opportunamente quali campi possono essere utili.

Per la gestione del carrello prevista nel punto (c), l'applicazione deve tenere traccia delle informazioni minimali necessarie per tenere traccia del prodotto, della quantità e per calcolare il totale complessivo. Ipotizza anche in questo caso l'organizzazione minimale dei dati che reputi necessaria per questa gestione, giustificandola opportunamente.

Le azioni visibili che l'applicazione deve permettere all'azienda per il punto (a) sono le seguenti:

- caricamento di un prodotto;
- visualizzazione dell'elenco dei prodotti;
- ricerca di un prodotto fornito il codice;
- caricamento dei metodi di pagamento che vengono resi disponibili agli acquirenti.

Le azioni visibili che l'applicazione deve permettere al cliente per il punto (b) sono le seguenti:

- registrazione da parte di un cliente al sito;
- scelta dei metodi di pagamento tra quelli resi disponibili dall'azienda o loro variazione;
- modifica dei propri dati: scegli opportunamente cosa permettere di modificare fornendo una motivata giustificazione.

Le azioni visibili che l'applicazione deve permettere all'acquirente per il punto (c) sono le seguenti:

- caricamento di un prodotto nel carrello;
- eliminazione di un prodotto dal carrello;
- visualizzazione del contenuto del carrello con il totale da pagare.

Documentazione di progetto finale

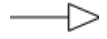
Andando avanti con lo sviluppo di applicazioni la documentazione di progetto diventerà sempre più una rappresentazione concettuale della realtà da astrarre e sarà sempre meno legata al codice prodotto. Questo è dovuto al fatto che le applicazioni diventeranno sempre più complesse e non sarà più utile documentare l'applicazione in ogni suo dettaglio. Diventerà utile invece una progettazione che individui le strutture dati utilizzate, le relazioni che intercorrono tra di loro e i casi d'uso dell'applicazione.

Le sezioni da prevedere sono le seguenti.

Obiettivo - In questa sezione deve essere riportato il problema proposto.

Progettazione strutture dati - In questa sezione devono essere descritte le strutture dati utilizzate, come array, struct, array di struct. Giustifica adeguatamente le scelte fatte.

Relazioni tra le entità (tra le strutture dati) - In questa sezione devono essere descritte le relazioni tra le entità coinvolte nell'applicazione, facendo anche riferimento alle strutture dati usate per rappresentarle. Possono essere utilizzati dei grafici che collegano le entità rappresentate da dei rettangoli. I collegamenti che finora sono stati usati sono:

- **Aggregazione**  Questa relazione prevede che un'entità abbia (HAS-A) un insieme di altre entità fungendo da contenitore per la collezione. I componenti contenuti possono esistere senza il contenitore e possono essere associati a diversi contenitori allo stesso tempo. Ad esempio, un cliente HA un'aggregazione di metodi di pagamento, ma l'eventuale eliminazione del cliente non provoca anche l'eliminazione del metodo di pagamento.
- **Composizione**  Una relazione di composizione si ha quando un'entità è composta da una o più istanze di un'altra entità. La differenza tra questa relazione e l'aggregazione è che il componente può esistere solo come parte del contenitore. Anche questa relazione è comunque una relazione di tipo HAS-A. Ad esempio, il sistema informativo di un rivenditore online di prodotti HA delle collezioni di prodotti, clienti e metodi di pagamento che sono delle sue parti e, se viene meno il sistema informativo aziendale le varie collezioni in esso contenute cessano di esistere con esso.
- **Generalizzazione/Specializzazione**  Una relazione di specializzazione si ha quando un'entità che rappresenta un concetto generico viene specializzata con l'aggiunta di ulteriori attributi, al fine di rappresentare una nuova entità che contiene sia le caratteristiche della generalizzazione, sia le nuove caratteristiche della specializzazione. Ad esempio, l'entità generica che rappresenta una persona può essere usata per rappresentare una parte delle caratteristiche di un insegnante. L'entità dell'insegnante ingloba tutte le caratteristiche di una persona perché È UNA (IS-A) persona, ma aggiunge anche ulteriori caratteristiche specifiche dell'insegnante, come ad esempio il numero di matricola o le abilitazioni all'insegnamento delle varie discipline.

Altre tipologie di relazioni che saranno introdotte quando sarà necessario sono:

- **Dipendenza** 

- Associazione 
- Generalizzazione 

Casi d'uso

Caso d'uso principale - In questa sezione si descrive come viene usata l'applicazione dal punto di vista di un utente che ha fatto una richiesta al sistema. Un caso d'uso cattura la sequenza di eventi *visibili* attraverso cui il sistema deve procedere per rispondere ad una singola richiesta dell'utente. I casi d'uso sono derivati dai colloqui effettuati con gli [stakeholder](#) e, per loro natura sono destinati a cambiare frequentemente, quindi è bene mantenerli semplici. Ogni caso d'uso presenta almeno un *caso principale* che descrive la risposta del sistema quando tutto procede correttamente. Per ogni *Caso d'uso principale* è spesso necessario prevedere anche degli esempi di ciò che deve essere visualizzato e inserito.

Casi d'uso alternativi - Spesso nei *Casi d'uso principali* sono presenti dei *Casi d'uso alternativi*, che si verificano quando il *Caso d'uso principale* può presentare dei problemi da gestire. Nei *Casi d'uso alternativi* è necessario indicare quando si verificano e cosa bisogna fare per recuperare la situazione anomala. Anche per i *Casi d'uso alternativi* si prevedono degli esempi di ciò che viene visualizzato dal programma.

Principi di progettazione usati - In questa sezione devono essere indicati e spiegati i principi di programmazione usati, come ad esempio

- la [programmazione modulare](#) per separare l'applicazione in moduli distinti;
- il [principio della separazione delle competenze](#) dei moduli (*SoC - Separation of Concern*);
- il [principio della singola responsabilità](#) delle funzioni (*SRP - Single Responsibility Principle*);
- l'attuazione del [principio di incapsulamento](#) delle (*Encapsulation principle*) delle strutture dati e delle funzioni nei moduli di competenza;
- il [principio Open-Close](#) (*Open-Closed principle*) sulle concetto di estensione del codice di un modulo invece della sua modifica.

Codice dell'applicazione - In questa sezione viene riportato il codice dell'applicazione, se richiesto.

Esempi di progettazione

Array paralleli - Città e PM10

Analisi presa dal lavoro iniziale dello studente Lorusso (a.s.2024/2025)

Obiettivo - Il seguente esempio permette il caricamento dei nomi delle città e dei valori di PM10 per ognuna di esse. L'applicazione, oltre al caricamento dei dati, permette di visualizzare i nomi delle città che hanno un valore di PM10 pari al massimo e di visualizzare genericamente lo stato di tutte le città.

Progettazione strutture dati - Vengono usati due array paralleli statici, uno per contenere i nomi delle città, `citta[NUM_CITTA][LUNGHEZZA]`, e l'altro per contenere il valore del PM10, `particolato[NUM_CITTA]`. La macro `NUM_CITTA` indica il numero massimo di città che possono essere caricate nell'array `citta` e i corrispondenti valori di PM10 nell'array `particolato`. La macro `LUNGHEZZA` serve definisce il numero massimo di caratteri per i nomi delle città.

Relazioni tra le entità - Ogni nome dell'array `citta` corrisponde ad un valore di PM10 nell'array `particolato` che occupano la stessa posizione nei due array.

Casi d'uso

Caso d'uso principale - menù

1. Il sistema visualizza il menù delle opzioni disponibili.
2. L'utente inserisce una scelta.
3. Svolta l'operazione scelta dall'utente, il menù viene nuovamente visualizzato, tranne nel caso in cui sia stata scelta l'uscita dall'applicazione che la fa terminare.

Esempio del menù:

- 1) Carica citta' con particolato
 - 2) Citta' con PM10 massimo
 - 3) Visualizza citta' e particolato
 - 4) Esci
- Scelta: _

Casi d'uso alternativi - menù

1. Se l'utente inserisce nella scelta un valore non valido, il programma visualizza un messaggio d'errore e ripropone la visualizzazione del menù.
2. Se l'utente inserisce come scelta un carattere al posto di un valore intero, il programma visualizza un messaggio d'errore e ripropone la visualizzazione del menù.
3. Se l'utente chiede di visualizzare le città con PM10 massimo (scelta 2) o le città con i corrispondenti particolati (scelta 3) prima di aver inserito le città e i propri PM10 (scelta 1), il programma visualizza un messaggio d'errore e ripropone il menù.

Casi d'uso principali - Inserimento città e PM10

1. Il programma per ogni città richiede l'inserimento prima el nome della città e poi del suo PM10
2. Viene riproposto il menù al termine dell'inserimento dei dai

Casi d'uso alternativi - Inserimento città e PM10

1. Se il PM10 è minore di 0 segnalare l'errore e richiedere l'input
2. Se per il PM10 venisse inserito un carattere segnalare l'errore e richiedere l'input

Casi d'uso principali - Visualizzazione dati

1. Il sistema deve visualizzare tutte le città con i corrispondenti valori di PM10 e successivamente visualizza nuovamente il menù

Casi d'uso principali - Visualizzazione delle posizioni massime

1. Il programma visualizza la città con i valore massimo di PM10 e successivamente visualizza nuovamente il menù

Principi di programmazione usati -

Codice dell'applicazione

main.c

```
/**
 * Città e PM10
 */
#include <stdio.h>
#include <stdlib.h>
#include "lib/menu.h"
#include "lib/citta.h"

#define NUM_CITTA 5
#define LUNGHEZZA 20
#define NUM_OPZIONI 4

int main() {
    char citta[NUM_CITTA][LUNGHEZZA];
    float particolato[NUM_CITTA];
    int scelta;
    char *opzioni = "1) Carica citta' con particolato\n"
                   "2) Citta' con PM10 massimo\n"
                   "3) Visualizza citta' e particolato\n"
                   "4) Esci\n"
                   "Scelta: ";

    scelta = menu(opzioni, NUM_OPZIONI);
    while (scelta != NUM_OPZIONI) {
        switch(scelta) {
            case 1:
```

```

        /*Caricamento delle singole città e del corrispondente
        valore di PM10*/
        caricaDati(NUM_CITTA, LUNGHEZZA, citta, particolato);
        break;
    case 2: ;
        /*Le due istruzioni commentate permettono di detrminare
        la posizione del singolo valore massimo, il primo che
        viene trovato, e di visualizzare il nome della città
        corrispondente.*/
        //posizione = posizioneMassimo(NUM_CITTA, particolato);
        //printf("La città' con il PM10 massimo a' %s\n", citta[posizione]);

        /*Le successive istruzioni permettono di determinare le
        posizioni di tutte le città che hanno un valore pari al
        massimo e di visualizzare tutti i loro nomi.*/
        int posizioniMax[NUM_CITTA]; //Array che contiene le posizioni
                                     //dei massimi
        //Funzione che salva in posizioniMax le posizioni dei massimi
        posizioniMassime(NUM_CITTA, particolato, posizioniMax);
        //Funzione che visualizza tutte e sole le città con i valori
        //massimi di PM10
        visualizzaMassimi(NUM_CITTA, LUNGHEZZA, citta, posizioniMax);
        break;
    case 3:
        //Visualizza tutti i nomi delle città con il corrispondente
        //valore di PM10
        visualizzaDati(NUM_CITTA, LUNGHEZZA, citta, particolato);
        break;
    }
    scelta = menu(opzioni, NUM_OPZIONI);
}
return 0;
}

```

menu.h

```

#pragma once
int menu(const char *opzioni, int numOpzioni);

```

menu.c

```

#include <stdio.h>
/**
 * Visualizza il menù e restituisce la scelta fatta dall'utente.
 * @param opzioni - Elenco di opzioni del menù
 * @param numOpzioni - Numero di opzioni presenti nel menù
 * @return la scelta dell'utente
 */
int menu(const char *opzioni, int numOpzioni) {
    int scelta;
    int inputOK = 0;

    printf("%s", opzioni);
    inputOK = scanf("%d", &scelta);
    while(getchar() != '\n');
    while(inputOK != 1 || (scelta < 1 || scelta > numOpzioni)) {
        printf("Errore di scelta\n");
        printf("%s", opzioni);
    }
}

```

```

        inputOK = scanf("%d", &scelta);
        while(getchar() != '\n');
    }
    return scelta;
}

```

citta.h

```

#pragma once
void caricaDati(int numCitta, int lunghezza, char citta[][lunghezza],
float particolare[]);
int posizioneMassimo(int numCitta, float particolare[]);
void visualizzaDati(int numCitta, int lunghezza, char citta[][lunghezza],
float particolare[]);
void posizioniMassime(int numCitta, float particolare[], int posizioniMax[]);
void visualizzaMassimi(int numCitta, int lunghezza, char citta[][lunghezza],
int posizioni[]);

```

citta.c

```

#include <stdio.h>

```

```

/**
 * Carica i nomi delle città e il corrispondente valore del PM10
 * @param numCitta - Numero di città
 * @param lunghezza - Lunghezza massima del nome di tutte le città
 * @param citta[][] - Nomi delle città
 * @param particolare[] - Valori di PM10 per ogni città
 */
void caricaDati(int numCitta,
                int lunghezza,
                char citta[][lunghezza],
                float particolare[]) {

    int inputOK; //Serve per verificare l'inserimento di un corretto
                //valore numerico
    //Ciclo di caricamento dei nomi delle città e del PM10
    //verificando che quest'ultimo sia corretto
    for(int i = 0; i < numCitta; ++i) {
        printf("Citta': ");
        scanf("%s", citta[i]);
        printf("PM10: ");
        inputOK = scanf("%f", &particolare[i]);
        while(getchar() != '\n');
        while(inputOK != 1 || particolare[i] < 0) {
            printf("Errore particolare\n");
            printf("PM10: ");
            inputOK = scanf("%f", &particolare[i]);
            while(getchar() != '\n');
        }
    }
}

/**
 * Individuazione della posizione del massimo.
 * @param numCitta - Numero di città
 * @param particolare[] - Valori di PM10 delle singole città
 * @return la posizione nell'array del primo valore massimo di

```

```
*          PM10 trovato nell'array
*/
int posizioneMassimo(int numCitta, float particolato[]) {
    int posMax = 0; //Posizione del primo valore massimo di PM10
                    //trovato nell'array
    float max = particolato[0]; //Valore massimo di PM10 presente
                               //nell'array
    //Ciclo per individuare la posizione del valore massimo di PM10
    for(int i = 1; i < numCitta; ++i) {
        if(particolato[i] > max) {
            max = particolato[i];
            posMax = i;
        }
    }
    //Restituzione della posizione del valore massimo del PM10
    return posMax;
}

/**
 * Visualizzazione dell'elenco delle città con il relativo
 * valore di PM10.
 * @param numCitta - Numero di città
 * @param lunghezza - Lunghezza massima del nome di tutte le città
 * @param citta[][] - Nomi delle città
 * @param particolato[] - Valori di PM10 per ogni città
 */
void visualizzaDati(int numCitta,
                    int lunghezza,
                    char citta[][lunghezza],
                    float particolato[]) {
    //Ciclo di visualizzazione delle città e del corrispondente PM10
    for(int i = 0; i < numCitta; ++i) {
        printf("%s: %f\n", citta[i], particolato[i]);
    }
}

/**
 * Individuazione di tutte le posizioni che contengono un
 * valore massimo di PM10.
 * @param numCitta - Numero di città
 * @param particolato[] - Valori di PM10 per ogni città
 * @param posizioniMax[] - Indici delle posizioni che corrispondono
 *                          ad un valore di PM10 pari al massimo
 */
void posizioniMassime(int numCitta,
                      float particolato[],
                      int posizioniMax[]) {
    //Ciclo di inizializzazione dell'array degli indici
    //delle posizioni dei massimi. Inizializzate a -1
    //in modo da avere un valore impossibile per un indice
    //da usare per terminare i futuri cicli senza dover
    //scorrere l'intero array
    for(int i = 0; i < numCitta; ++i) {
        posizioniMax[i] = -1;
    }
}
```

```
//Individuazione della posizione del primo valore massimo di PM10
int posMax = posizioneMassimo(numCitta, particolato);
//Memorizzazione del valore massimo di PM10 da usare per i confronti
float max = particolato[posMax];
//Ciclo di individuazione e salvataggio delle posizioni dei valori
//pari al massimo di PM10
for(int i = 0, j = 0; i < numCitta; ++i) {
    if(particolato[i] == max) {
        //In questa istruzione bisogna usare necessariamente j++
        //e non ++j perché in questo modo prima viene usato il
        //valore di j per indicizzare la posizione in posizioniMax
        //e dopo viene incrementato il valore di j.
        //Se si usasse ++j prima verrebbe incrementato il valore di j
        //e poi questo sarebbe usato per indicizzare la posizione in
        //posizioniMax.
        posizioniMax[j++] = i;
    }
}

/**
 * Visualizzazione di tutte le città che hanno un PM10 pari al massimo.
 * @param numCitta - Numero di città
 * @param lunghezza - Lunghezza massima del nome di tutte le città
 * @param citta[][] - Nomi delle città
 * @param posizioni[] - Indici delle posizioni che corrispondono
 *                     ad un valore di PM10 pari al massimo
 */
void visualizzaMassimi(int numCitta,
                      int lunghezza,
                      char citta[][lunghezza],
                      int posizioni[]) {
    //Ciclo di visualizzazione
    for(int i = 0; i < numCitta; ++i) {
        //Appena viene trovato nell'array posizioni[] un valore pari a -1
        //si è certi che non sono presenti altri indici validi
        if(posizioni[i] == -1) {
            break;
        } else {
            //Viene visualizzata la città che si trova all'indice indicato
            //dalla i-esima posizione di posizioni[]
            int pos = posizioni[i];
            printf("La città' con il PM10 massimo a' %s\n", citta[pos]);
        }
    }
}
```

Struct - Vendita di prodotti

Problema

Una nota azienda di vendita online ha necessità di migliorare la gestione e la vendita dei suoi prodotti. A tal proposito l'azienda ti commissiona un software che gli permetta di gestire l'elenco dei prodotti, dei clienti e delle modalità di pagamento che i clienti possono usare.

Per il caricamento della lista dei prodotti l'azienda deve prevedere il caricamento del codice identificativo, la descrizione e il prezzo unitario. Inoltre l'azienda vuole avere la possibilità di aggiungere un nuovo prodotto alla lista e di visualizzare l'elenco di tutti i prodotti.

Di ogni cliente che si registra l'azienda vuole memorizzare tutte le informazioni anagrafiche e la lista delle modalità di pagamento. Il software deve quindi prevedere la registrazione di un nuovo cliente e la visualizzazione di tutti i clienti.

Le modalità di pagamento usate dal cliente possono essere molteplici e modificabili secondo il desiderio del cliente stesso, ma è l'azienda che decide quali modalità di pagamento rende disponibili ai clienti.

Progettazione strutture dati

L'azienda di vendita online necessita di diverse strutture dati per gestire i prodotti, i clienti e i metodi di pagamento. A tale scopo si decide di strutturare le diverse entità al fine di usarle per tipizzare una collezione di liste che costituirà il sistema informativo dell'azienda.

Dalle specifiche raccolte i prodotti sono rappresentabili dalla seguente struttura:

```
struct Prodotto {  
    char codice[LEN_COD];  
    char descrizione[LEN_DES];  
    float prezzo;  
}
```

Questa struttura viene usata per tipizzare un array di prodotti atto a contenere l'elenco di tutti i prodotti venduti dall'azienda:

```
struct Prodotto prodotti[DIM_PRODOTTI];
```

I colloqui informativi con l'azienda indicano che i metodi di pagamento sono gestiti internamente dall'azienda in base a convenzioni stipulate con diversi operatori finanziari e bancari. Prescindendo dalle tecniche di autenticazione richieste per il loro uso, che vengono demandate alle singole società finanziarie, l'elenco dei metodi di pagamento viene collezionato in un array di stringhe le cui posizioni devono essere sufficientemente ampie per contenere la dicitura dei vari metodi.

```
char metodiPagamento[DIM_METODI][LEN_METODI];
```

La scelta di confinare i metodi di pagamento in una struttura dati separata permette di perseguire il principio *incapsula ciò che varia* (*encapsulate what varies*) disaccoppiando la struttura di gestione dei metodi di pagamento da quella dei clienti, permettendo di variarle in modo indipendente, sia sui contenuti, sia sulla struttura.

Di conseguenza per i clienti l'azienda specifica che vuole memorizzare nel suo sistema informativo sia i dati anagrafici, sia i metodi di pagamento che il cliente decide di scegliere tra quelli resi disponibili dall'azienda stessa.

Avendo applicato il principio di *incapsula ciò che varia*, è possibile creare una struttura che raccolga i dati anagrafici del cliente quando si registreranno, con una aggregazione di metodi di pagamento formalizzabile come un array di interi che indicano la posizione del metodo nell'array `metodiPagamento[ ] [ ]`:

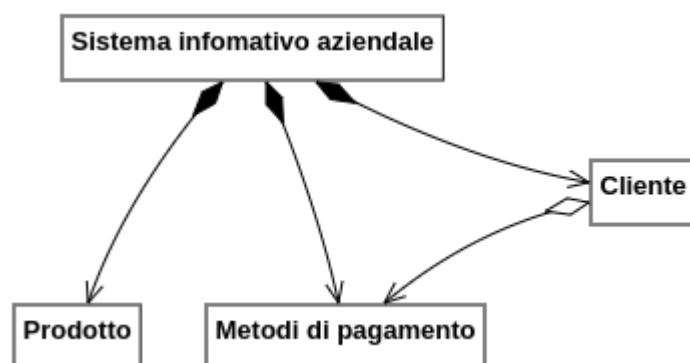
```
struct Cliente {  
    char nome[LEN_NOMINATIVO];  
    char cognome[LEN_NOMINATIVO];  
    char cellulare[LEN_CELL];  
    char indirizzo[LEN_INDIRIZZO];  
    char email[LEN_EMAIL];  
    int metodiPagamentoCliente[DIM_METODI];  
};
```

Usando questa struttura è possibile tipizzare un array dei clienti che diventerà parte integrante del sistema informativo aziendale:

```
struct Cliente clienti[DIM_CLIENTI];
```

Relazioni tra le entità

Considerando il sistema informativo dell'azienda nel suo complesso è possibile rappresentarlo graficamente nel seguente modo.



Il sistema informativo è una composizione di prodotti, clienti e metodi di pagamento, rappresentate dal simbolo . Questo vuol dire che il sistema informativo HA delle collezioni di prodotti, clienti e metodi di pagamento che sono delle sue parti e, se viene meno il sistema informativo aziendale le varie collezioni cessano di esistere con esso.

Invece il cliente HA un'aggregazione di metodi di pagamento, rappresentata dal simbolo  che rappresenta una sua parte, ma l'eventuale eliminazione del cliente non provoca anche l'eliminazione del metodo di pagamento.

Casi d'uso

I casi d'uso descrivono il punto di vista di un utente che ha fatto una richiesta al sistema. Un caso d'uso cattura la sequenza di eventi *visibili* attraverso cui il sistema deve procedere per rispondere ad una singola richiesta dell'utente.

Tutti i casi d'uso esposti in questa fase di sviluppo sono dal punto di vista dell'azienda, per come si aspetta che avvenga il caricamento e la consultazione dei dati. Non sono ancora contemplate altre casistiche che saranno sviluppate in futuro.

I casi d'uso esposti derivano dai colloqui effettuati con gli [stakeholder](#) e per loro natura sono destinati a cambiare frequentemente, quindi è bene mantenerli semplici.

Ogni caso d'uso presenta un *caso principale* che descrive la risposta del sistema quando tutto procede correttamente. Alle volte sono presenti dei *casi alternativi* quando il *caso principale* può presentare dei problemi da gestire.

Menù principale - Caso principale

- Viene visualizzato il menù principale per l'inserimento di prodotti, clienti e metodi di pagamento. Una delle scelte deve permettere la chiusura dell'applicazione.
- Il menù principale deve essere riproposto fino a quando non viene scelta l'opzione di chiusura dell'applicazione.

Menù principale - Casi alternativi

- Se l'utente inserisce una scelta non presente nel menù principale il sistema segnala l'errore e ripropone il menù.

Caricamento prodotto - Caso principale

- L'utente sceglie dal menù principale di caricare un prodotto.
- Inserisce i dati richiesti.
- Terminato l'inserimento viene visualizzato nuovamente il menù principale.

Caricamento prodotto - Casi alternativi

- *Codice prodotto già esistente*
 - Se il codice del prodotto è già presente nel sistema viene segnalato visualizzando anche i dati del prodotto corrispondente, riproponendo successivamente il menù principale.
[NOTA - L'attuale versione dell'applicazione non prevede la modifica di un prodotto, quindi non deve essere prevista nella progettazione e nello sviluppo]
- *Prezzo errato*
 - Se il prezzo unitario del prodotto inserito è minore o pari a zero il sistema deve segnalarlo richiedendo l'inserimento.
 - Se l'errore sul prezzo viene ripetuto per tre volte il sistema segnala il fallimento del caricamento del prodotto e visualizza il menù principale.

Visualizzazione prodotti - Caso principale

- L'utente sceglie dal menù principale l'opzione di visualizzazione dei prodotti e questi vengono visualizzati in formato tabellare.

Caricamento metodi di pagamento - Caso principale

- L'utente sceglie dal menù principale di caricare un nuovo metodo di pagamento.
- Il sistema visualizza l'elenco dei metodi di pagamento già presenti.
- L'utente inserisce i dati richiesti.
- Terminato l'inserimento viene visualizzato nuovamente il menù principale.

Caricamento metodi di pagamento - Casi alternativi

- Se il metodo di pagamento è già presente il sistema deve segnalarlo e tornare al menù principale.

Visualizzazione metodi di pagamento - Caso principale

- L'utente sceglie dal menù principale l'opzione di visualizzazione dei metodi di pagamento e questi vengono visualizzati come elenco.

Caricamento dei clienti - Caso principale

- Il cliente accede al sistema di registrazione.
[NOTA - La versione attuale dell'applicazione non prevede ancora lo sviluppo di questa fase che può essere simulata con dei dati fittizi].
- Inserisce i dati anagrafici richiesti.
- Sceglie da una lista i metodi di pagamento che desidera utilizzare.
- Terminato l'inserimento riceverà una email di conferma di registrazione.

Caricamento dei clienti - Casi alternativi

- *Numero di telefono errato*
 - Se il numero di telefono contenesse elementi non numerici l'applicazione non deve permettere la registrazione del cliente.
- *Email errata*
 - Se il formato dell'indirizzo email risulta sbagliato (*nome_utente@dominio*) l'applicazione non deve permettere la registrazione del cliente.
- *Metodo di pagamento mancante*
 - Se il cliente non sceglie almeno un metodo di pagamento l'applicazione non deve permettere la registrazione del cliente.

Visualizzazione clienti - Caso principale

- L'utente sceglie dal menù principale l'opzione di visualizzazione dei clienti e questi vengono visualizzati in forma tabellare seguendo l'ordine alfabetico del cognome.

Complessità degli algoritmi

Vedere questi video:

- [Big O Notation - Full Course](#)
- [Learn Big O Notation In 12 Minutes](#)
- [Big O: How Code Slows as Data Grows](#)

C Language Manuals

- [GNU C Language Manual](#) by [GNU Operating System](#)
- [GNU Coding Standards](#) by [GNU Operating System](#)
- [C Programming Tutorials](#) by [Portfolio Courses](#)
- [C Programming Examples](#) by [Portfolio Courses](#)
- [C Programming Full Course for free](#) by [Bro Code](#), [YouTube](#)
- [Informatica \(12BHD\) 2011/2012](#) by [Fulvio Corno](#), [YouTube](#)
- [C Style: Standards and Guidelines](#), [Syque](#)

Per le OII

[How to use assertions in C](#) by [John Reekie](#)

[Assert in C Programming](#) by [javaTpoint](#)

OOP in C language

Tutorial

- [Chapter 15: Navigating Object-Oriented Programming in C](#) by [TheCloudStrap.com](#)
- [Object-oriented Programming with ANSI-C](#) by [Axel-Tobias Schreiner](#)
- [C Language: #ifndef Directive](#) by [Tech on the Net](#)

Appendici

A - C course videos

1. [How To Setup A C Programming Environment On Windows](#)
2. [How To Setup A C Programming Environment On A Mac](#)
3. [Basics of variables and types | C Programming Tutorial](#)
4. [printf Basics | C Programming Tutorial](#)
5. [scanf Basics | C Programming Tutorial](#)
6. [if statements | C Programming Tutorial](#)
7. [Logical Operators | C Programming Tutorial](#)
8. [switch Statements | C Programming Tutorial](#)
9. [while loop C | Programming Tutorial](#)
10. [do while loop | C Programming Tutorial](#)
11. [for loops | C Programming Tutorial](#)
12. [break and continue statements | C Programming Tutorial](#)
13. [Function Basics | C Programming Tutorial](#)
14. [Array Basics | C Programming Tutorial](#)
15. [String Basics | C Programming Tutorial](#)
16. [Type Casting | C Programming Tutorial](#)
17. [File Access Basics | C Programming Tutorial](#)
18. [Command Line Arguments | C Programming Tutorial](#)
19. [Recursion | C Programming Tutorial](#)
20. [Constants | C Programming Tutorial](#)
21. [typedef Basics | C Programming Tutorial](#)
22. [struct Basics | C Programming Tutorial](#)
23. [Introduction to Pointers | C Programming Tutorial](#) ← **IMPORTANTE!!!** Per ora puoi fermarti dopo la spiegazione della funzione `swap()`, al minuto 18:23, la funzione `malloc()` la vedremo più avanti
24. [Pass By Reference | C Programming Tutorial](#)
25. [Passing an Array to a Function | C Programming Tutorial](#)
26. [Dynamic Memory Allocation | C Programming Tutorial](#)
27. [malloc vs calloc Differences Explained | C Programming Tutorial](#)
28. [realloc Basics | C Programming Tutorial](#)
29. [realloc Security Vulnerability | C Programming Tutorial](#)
30. [2D Array Basics | C Programming Tutorial](#)
31. [Heap Inspection Security Vulnerability | C Programming Tutorial](#)
32. [postfix vs. prefix increment operator | C Programming Tutorial](#)
33. [ctype Library | C Programming Tutorial](#)
34. [Variable Scope | C Programming Tutorial](#)
35. [Pointer Notation | C Programming Tutorial](#)
36. [Main Function Return Value | C Programming Tutorial](#)
37. [Comments | C Programming Tutorial](#)
38. [strcmp\(\) function | C Programming Tutorial](#)
39. [strpbrk\(\) function | C Programming Tutorial](#)
40. [strchr\(\) function | C Programming Tutorial](#)
41. [strstr\(\) function | C Programming Tutorial](#)
42. [strtok\(\) function | C Programming Tutorial](#)
43. [strcspn\(\) function | C Programming Tutorial](#)
44. [strchr\(\) function | C Programming Tutorial](#)
45. [strspn\(\) function | C Programming Tutorial](#)

46. [strcat\(\) and strncat\(\) functions | C Programming Tutorial](#)
47. [Handling String Input With Spaces | C Programming Tutorial](#)
48. [enum | C Programming Tutorial](#)
49. [Ternary Operator | C Programming Tutorial](#)
50. [memcpy\(\) function | C Programming Tutorial](#)
51. [memset\(\) function | C Programming Tutorial](#)
52. [exit\(\) function | C Programming Tutorial](#)
53. [Time Library | C Programming Tutorial](#)
54. [sleep\(\) and usleep\(\) functions | C Programming Tutorial](#)
55. [How True and False Boolean Values Work In C](#)
56. [Relational Operators | C Programming Tutorial](#)
57. [Arithmetic Operations | C Programming Tutorial](#)
58. [Assignment Operators | C Programming Tutorial](#)
59. [div\(\) function | C Programming Tutorial](#)
60. [modf\(\) function | C Programming Tutorial](#)
61. [Error Handling | C Programming Tutorial](#) *usa i file*
62. [fmod\(\) function | C Programming Tutorial](#)
63. [round\(\), ceil\(\), floor\(\) and trunc\(\) double rounding functions | C Programming Tutorial](#)
64. [How to create functions with a variable number of arguments using stdarg.h | C Programming Tutorial](#)
65. [pow\(\) function | C Programming Tutorial](#)
66. [Random Number Generation | C Programming Tutorial](#)
67. [Bitwise Operators | C Programming Tutorial](#)
68. [remove\(\) function for deleting files | C Programming Tutorial](#)
69. [rename\(\) Function For Renaming Files | C Programming Tutorial](#)
70. [Static Local Variables | C Programming Tutorial](#) *usa i file*
71. [strcpy\(\) and strncpy\(\) functions | C Programming Tutorial](#)
72. [Arrow Operator For Pointers To Structs | C Programming Tutorial](#)
73. [goto Statements | C Programming Tutorial](#)
74. [rewind\(\) function to reset file pointer to file start | C Programming Tutorial](#)
75. [sprintf\(\) Function | C Programming Tutorial](#)
76. [getenv\(\) Function To Get The Value Of An Environment Variable | C Programming Tutorial](#)
77. [system\(\) Function To Run Shell Commands | C Programming Tutorial](#)
78. [strtod\(\) Converts String To Double And Returns Remaining String Portion | C Programming Tutorial](#)
79. [Convert A String To A Double, Int Or Long Int With atof\(\), atoi\(\) or atol\(\) | C Programming Tutorial](#)
80. [How To Use atexit\(\) To Set A Function To Run On Program Exit | C Programming Tutorial](#)
81. [abort\(\) Function | C Programming Tutorial](#)
82. [Using assert\(\) To Help Debug Our Program With Assertions | C Programming Tutorial](#)
83. [memccpy\(\) Function | C Programming Tutorial](#)
84. [memchr\(\) Function | C Programming Tutorial](#)
85. [tmpnam\(\) Function To Create Temporary Filenames | C Programming Tutorial](#)
86. [fseek\(\) Function | C Programming Tutorial](#)
87. [Union Data Types | C Programming Tutorial](#)

88. [How To Create A Library And Split A Program Across Multiple Files | C Programming Tutorial](#)
89. [Static Global Variables | C Programming Tutorial](#)
90. [Binary File Access Introduction | C Programming Example](#)
91. [Register Variables | C Programming Tutorial](#)
92. [Introduction To Threads \(pthreads\) | C Programming Tutorial](#)
93. [Function Pointers | C Programming Tutorial](#)
94. [Race Conditions Explained With An Example | C Programming Tutorial](#)
95. [Make A File Directory With mkdir\(\) | C Programming Tutorial](#)
96. [lvalues and rvalues | C Programming Tutorial](#)
97. [Mutex Introduction \(pthreads\) | C Programming Tutorial](#)
98. [clearerr\(\) Function | C Programming Tutorial](#)
99. [ungetc\(\) Function | C Programming Tutorial](#)
100. [Detached Threads \(pthreads\) | C Programming Tutorial](#)
101. [Comma Operator \(And When To Use It\) | C Programming Tutorial](#)
102. [truncate\(\) Function To Truncate A File \(POSIX Library\) | C Programming Tutorial](#)
103. [Memory Leaks And How To Prevent Them | C Programming Tutorial](#)
104. [fprintf\(\) Function Deep Dive | C Programming Tutorial](#)
105. [fputs\(\) Function | C Programming Tutorial](#)
106. [int main\(\) vs int main\(void\) | C Programming Tutorial](#)
107. [Why We Should NOT Use void main\(\) | C Programming Tutorial](#)
108. [Why We Should Never Use gets\(\)... And Why To Use fgets\(\) Instead | C Programming Tutorial](#)
109. [Function like Macros | C Programming Tutorial](#) *usa gli array*
110. [#ifdef #ifndef Conditional Compilation Directives | C Programming Tutorial](#)
111. [#if #elif #else Conditional Compilation Directives | C Programming Tutorial](#)
112. [#error Preprocessor Directive | C Programming Tutorial](#)
113. [Token-Pasting Operator ## | C Programming Tutorial](#)
114. [Predefined Preprocessor Macros + Log File Use Case | C Programming Tutorial](#)
115. [String In Char Array VS. Pointer To String Literal | C Programming Tutorial](#)
116. [Constant Pointer VS. Pointer To A Constant | C Programming Tutorial](#)
117. [Make A Directory Using mkdir\(\) On Windows | C Programming Tutorial](#)
118. [Get The Current Working Directory Using getcwd\(\) On Windows | C Programming Tutorial](#)
119. [size_t Type | C Programming Tutorial](#)
120. [Change The Current Working Directory Using chdir\(\) On Windows | C Programming Tutorial](#)
121. [Function Parameters VS. Arguments | C Programming Tutorial](#)
122. [memmove\(\) Function | C Programming Tutorial](#)
123. [sizeof Operator | C Programming Tutorial](#) *usa la malloc()*
124. [Nested Functions | C Programming Tutorial](#)
125. [Find Middle Index Without Integer Overflow | C Programming Tutorial](#)
126. [array vs &array Pointers Difference Explained | C Programming Tutorial](#) *usa le matrici 2D*
127. [function vs. &function Pointers | C Programming Tutorial](#)
128. [Dynamically Allocate A 2D Array | C Programming Tutorial](#)
129. [Increase The Size Of A Dynamically Allocated 2D Array | C Programming Tutorial](#)
130. [Difference Between %f And %lf Format Specifiers | C Programming Tutorial](#)

- ~~131. [int *p vs int* p Pointer Declarations | C Programming Tutorial](#)~~
- ~~132. [Convert A For Loop To A While Loop | C Programming Tutorial](#)~~
- ~~133. [How To Return An Array From A Function | C Programming Tutorial](#)~~ ←
IMPORTANTE!!! Per ora puoi fermarti al minuto 8:03, la funzione malloc() la vedremo più avanti
- ~~134. [Return A Dynamically Allocated 2D Array From A Function | C Programming Tutorial](#)~~
- ~~135. [Static Functions | C Programming Tutorial](#)~~
- ~~136. [Find The Length Of An Array Using sizeof\(\) | C Programming Tutorial](#)~~ ←
IMPORTANTE!!! Per ora puoi fermarti al minuto 13:54, la funzione malloc() la vedremo più avanti
- ~~137. [If Statement Best Practice | C Programming Tutorial](#)~~
- ~~138. [Dynamically Allocate An Array Of Structs | C Programming Tutorial](#)~~
- ~~139. [How To Return A String From A Function | C Programming Tutorial](#)~~ ←
IMPORTANTE!!! Per ora puoi fermarti al minuto 9:38, la funzione malloc() la vedremo più avanti
- ~~140. [Return Statement In A Void Function | C Programming Tutorial](#)~~
- ~~141. [Absolute Value Functions abs\(\) And fabs\(\) | C Programming Tutorial](#)~~
- ~~142. [Find A Square Root Using sqrt\(\) sqrtf\(\) sqrtl\(\) | C Programming Tutorial](#)~~
- ~~143. [NULL Pointer | C Programming Tutorial](#)~~
- ~~144. [Local vs. Global Variables | C Programming Tutorial](#)~~
- ~~145. [Floating Point Number Classification Functions | C Programming Tutorial](#)~~
- ~~146. [Structure Padding | C Programming Tutorial](#)~~
- ~~147. [How To Clear The Input Buffer | C Programming Tutorial](#)~~
- ~~148. [#line Preprocessor Directive | C Programming Tutorial](#)~~
- ~~149. [puts\(\) Function | C Programming Tutorial](#)~~
- ~~150. [Interleave Switch Cases Within Other Control Structures | C Programming Tutorial](#)~~
- ~~151. [Single vs. Multiple Recursion | C Programming Tutorial](#)~~
- ~~152. [Stack Overflow Errors | C Programming Tutorial](#)~~
- ~~153. [Direct vs. Indirect Recursion | C Programming Tutorial](#)~~
- ~~154. [Head vs Tail Recursion | C Programming Tutorial](#)~~
- ~~155. [tmpfile\(\) Function | C Programming Tutorial](#)~~
- ~~156. [Include Guards | C Programming Tutorial](#)~~ ← l'argomento è spiegato usando le struct che vedremo più avanti, ma cerca di comprendere come usare gli include guards, senza preoccuparti della complessità del codice
- ~~157. [tolower\(\) Function | C Programming Tutorial](#)~~
- ~~158. [How To Use #if 0 ... #endif To Remove Code For Debugging | C Programming Tutorial](#)~~
- ~~159. [isalpha\(\) Function | C Programming Tutorial](#)~~
- ~~160. [isdigit\(\) Function | C Programming Tutorial](#)~~
- ~~161. [isalnum\(\) Function | C Programming Tutorial](#)~~
- ~~162. [ispunct\(\) Function | C Programming Tutorial](#)~~
- ~~163. [isspace\(\) Function | C Programming Tutorial](#)~~
- ~~164. [How To Print The % Character Using printf\(\) | C Programming Tutorial](#)~~
- ~~165. [feof\(\) Function | C Programming Tutorial](#)~~
- ~~166. [isprint\(\) Function | C Programming Tutorial](#)~~
- ~~167. [iscntrl\(\) Function | C Programming Tutorial](#)~~
- ~~168. [Standard Error Stream \(stderr\) | C Programming Tutorial](#)~~
- ~~169. [isgraph\(\) Function | C Programming Tutorial](#)~~

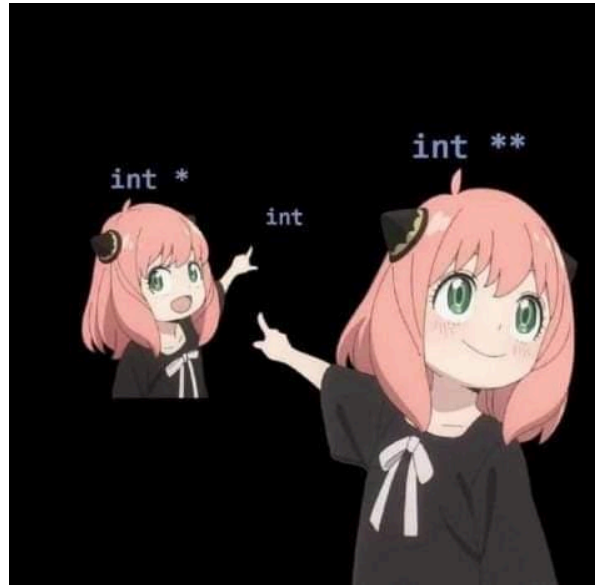
- 170. [Void Pointers | C Programming Tutorial](#)
- 171. [Designated Initializers For Arrays | C Programming Tutorial](#)
- 172. [Flexible Array Members | C Programming Tutorial](#)
- 173. [Null Terminator | C Programming Tutorial](#)
- 174. [_Bool Type Explained | C Programming Tutorial](#)
- 175. [Null Terminated String Safety Issues | C Programming Tutorial](#)
- 176. [strlen\(\) String Length Function \(POSIX\) | C Programming Tutorial](#)
- 177. [Dangling Pointers | C Programming Tutorial](#)
- 178. [do {...} while\(0\) In Macros Pattern | C Programming Tutorial](#)
- 179. [Multiline String Literals | C Programming Tutorial](#)
- 180. [Do we need to typecast the return value of malloc\(\)? | C Programming Tutorial](#)
- 181. [Declare One Variable Per Line Coding Style Advantages | C Programming Tutorial](#)
- 182. [time\(NULL\) Explained | C Programming Tutorial](#)
- 183. [Resize A String Stored In Dynamically Allocated Memory | C Programming Tutorial](#)
- 184. [How To Resize An Array | C Programming Tutorial](#)
- 185. [Void Functions | C Programming Tutorial](#)
- 186. [Legal Ways To Call Void Functions | C Programming Tutorial](#)
- 187. [Segmentation Faults And Common Causes | C Programming Tutorial](#)

B - Restituzione dell'indirizzo di memoria di un array da una funzione

L'applicazione si occupa di gestire l'input di un numero arbitrario di valori float, li ordina in ordine crescente e li visualizza.

L'architettura generale dell'applicazione è identica alla versione precedente, così come l'organizzazione delle funzioni in una libreria separata per rendere modulare l'applicazione stessa.

La differenza principale risiede nel fatto che in questa versione la funzione di allocazione dell'array prevede esplicitamente il passaggio del parametro attuale per indirizzo, cioè alla funzione viene passato l'indirizzo di memoria della variabile `float *array`; dichiarata nel `main()` e non il suo contenuto. Le altre funzioni continuano ad avere il passaggio dei parametri per valore.



- Prototipo funzione: `bool allocaArrayFloat(float **array, int n);`
Invocazione funzione: `int isAllocated = allocaArrayFloat(&array, n);`

Il parametro formale `float **array` prevede due referenziazione perché la funzione prevede il passaggio del parametro per indirizzo e, ciò che contiene il parametro attuale è a sua volta un indirizzo, cioè è un puntatore.

Nella variabile `array` all'interno della funzione `main()` deve essere salvato l'indirizzo di memoria della prima cella dell'array creato nello Heap dalla funzione `malloc()`. Perché ciò avvenga è necessario passare il parametro attuale `array` per indirizzo. Dato che però questa è una variabile puntatore a float, nella firma della funzione bisogna indicare entrambe le due referenziazioni, sia il passaggio per indirizzo del parametro, sia il fatto che è una variabile puntatore. Da questo deriva la doppia referenziazione nella firma della funzione.

All'interno della funzione l'accesso al parametro formale avviene sempre tramite l'operatore di dereferenziazione `*` in quanto il valore salvato o il controllo sul valore salvato devono essere fatti all'indirizzo di memoria contenuto nel parametro formale, come avviene normalmente quando si usa il passaggio dei parametri per indirizzo.

La funzione restituisce un valore booleano per lasciare la gestione dell'eventuale errore della mancata allocazione al `main()`, invece di gestirla internamente. Inoltre la funzione riceve come ulteriore parametro formale la dimensione dell'array per permetterne il suo dimensionamento dinamico.

Si osservi infine che la **modifica apportata alla funzione** `allocaArrayFloat()` **risulta completamente trasparente** nell'applicazione proprio per l'applicazione dei principi di **programmazione** visti finora: *separazione delle responsabilità* tra i moduli e le funzioni, *singola responsabilità* assegnata ad ogni modulo e ad ogni funzione, *basso accoppiamento*

tra i moduli e le funzioni, *alta coesione* nei moduli e nelle funzioni. La cosa importante è stata mantenere inalterata l'interfaccia pubblica della funzione dichiarata nello header `arraymanage.h`, in questo modo qualsiasi variazione venga applicata alle funzioni di gestione dell'array non deve richiedere una conseguente variazione nel `main()`. Si lascia come esercizio l'applicazione di un diverso algoritmo di ordinamento del contenuto dell'array, verificando che l'uso del nuovo algoritmo non influenzi in alcun modo il funzionamento generale dell'applicazione.

main.c

```
#include "lib/arraymanage/arraymanage.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
    int n;
    float *array;

    // Richiedi all'utente la dimensione dell'array
    printf("Inserisci la dimensione dell'array: ");
    scanf("%d", &n);

    // Alloca dinamicamente l'array di float
    int isAllocated = allocaArrayFloat(&array, n);
    if(!isAllocated) {
        printf("Errore: memoria non disponibile.");
        exit(1); // Esci con codice di errore 1
    }

    // Carica i valori nell'array
    caricaValori(array, n);

    // Stampa i valori inseriti nell'array
    stampaValori(array, n);

    // Ordina l'array in ordine crescente
    bubbleSort(array, n);

    // Stampa i valori inseriti nell'array
    stampaValori(array, n);

    // Dealloca l'array dinamico
    deallocaArrayFloat(array);

    return 0;
}
```

arraymanage.h

```
bool allocaArrayFloat(float **array, int n);
void caricaValori(float *array, int n);
```

```
void stampaValori(const float *array, int n);
void bubbleSort(float *array, int n);
void deallocaArrayFloat(float *array);
```

arraymanage.c

```
#ifndef INCLUDE_H
#define INCLUDE_H
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#endif // INCLUDE_H

/**
 * Alloca nello heap uno spazio di memoria pari a n volte la dimensione di un float
 * @param n numero di celle da allocare
 * @return il puntatore all'array allocato
 */
bool allocaArrayFloat(float **array, int n) {
    // Alloca dinamicamente un array di float di dimensione n
    *array = (float *)malloc(n * sizeof(float));

    // Verifica se l'allocazione è avvenuta correttamente
    if (*array == NULL) {
        return false;
    }

    return array;
}

/**
 * Carica i valori nell'array
 * @param *array puntatore all'array da caricare
 * @param n dimensione dell'array
 */
void caricaValori(float *array, int n) {
    // Inserisci i valori nell'array
    printf("Inserisci %d valori nell'array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%f", &array[i]);
    }
}

/**
 * Visualizza il contenuto dell'array
 * @param *array puntatore all'array da visualizzare
 * @param n dimensione dell'array
 */
void stampaValori(const float *array, int n) {
    // Stampa i valori inseriti nell'array
    printf("Valori nell'array:\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("%.2f ", array[i]);
    }
    printf("\n");
}

/**
 * Ordinamento crescente dell'array
 * @param *array puntatore all'array da ordinare
 * @param n dimensione dell'array
 */
void bubbleSort(float *array, int n) {
    bool swapped;
    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                float temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
            swapped = true;
        }
        // If there is no swap, then break
        if (swapped == false)
            break;
    }
}

/**
 * Deallocazione dell'array
 * @param *array puntatore all'array da deallocare
 */
void deallocaArrayFloat(float *array) {
    // Libera la memoria allocata per l'array
    free(array);
}
```

Una possibile esecuzione dell'applicazione genera i seguenti output a fronte degli input forniti.

Inserisci la dimensione dell'array: 5

Inserisci 5 valori nell'array:

7

2

4

9

8

Valori nell'array:

7.00 2.00 4.00 9.00 8.00

Valori nell'array:

2.00 4.00 7.00 8.00 9.00

C - Passaggio dei parametri per valore e per indirizzo

Uso dei puntatori nelle funzioni per la gestione dell'array di int

Testo

Caricare una collezione di valori interi, calcolare la media e visualizzare la collezione di valori e la media.

Macroanalisi

L'applicazione ha lo scopo di testare una serie di funzioni che saranno utili per i futuri progetti. Questa applicazione prevede il caricamento di una collezione di valori interi e la successiva visualizzazione del suo contenuto e del valore medio.

Microanalisi

L'applicazione prevede l'uso di funzioni separate per la gestione dei valori della collezione, invocate all'interno del main(). La prima funzione prevede il caricamento dei valori interi all'interno dell'array passato come parametro, successivamente sarà invocata la funzione per il calcolo della media, valore restituito dalla funzione stessa e, infine, viene invocata una funzione per la visualizzazione del contenuto della collezione. Il valore medio viene direttamente visualizzato all'interno della funzione main().

Tabella delle variabili e delle macro o costanti globali

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
ROWS	macro	Dimensione massima dell'array di interi	L	Inizializzata a 5

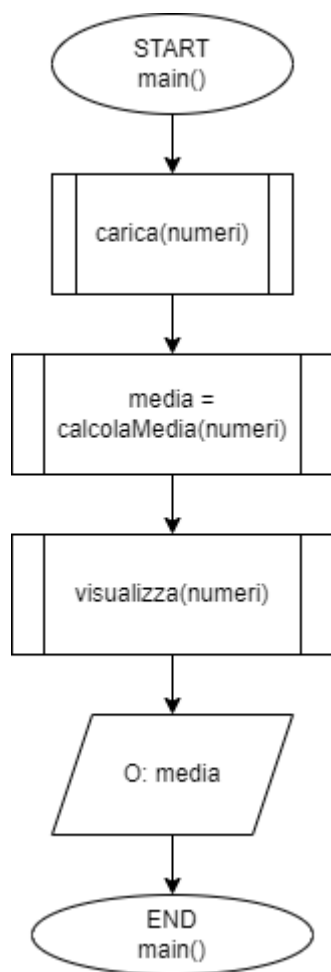
Funzione main()

La funzione si occupa di invocare la funzione di caricamento, la funzione per il calcolo della media e la funzione per la visualizzazione dei valori dell'array. Come ultima operazione visualizza il valore medio.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
numeri[ROWS]	int	Array di valori interi	I/O	-
media	float	Media dei valori contenuti nell'array	O	-

Flow-chart



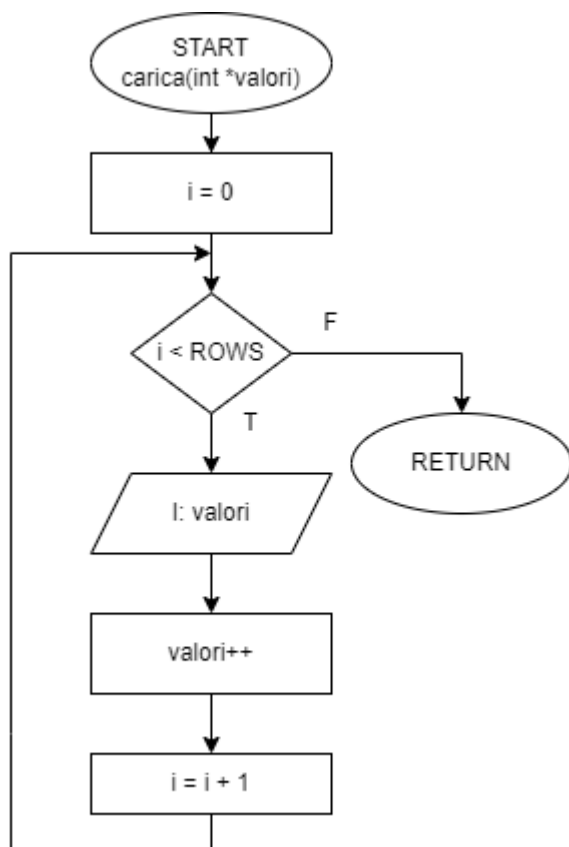
Funzione carica()

La funzione effettua il caricamento dei valori della collezione memorizzandoli nell'array passato come parametro.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
valori	int*	Puntatore all'array di valori interi. Parametro formale	I	-
i	int	Contatore del ciclo di caricamento	L	-

Flow-chart carica()

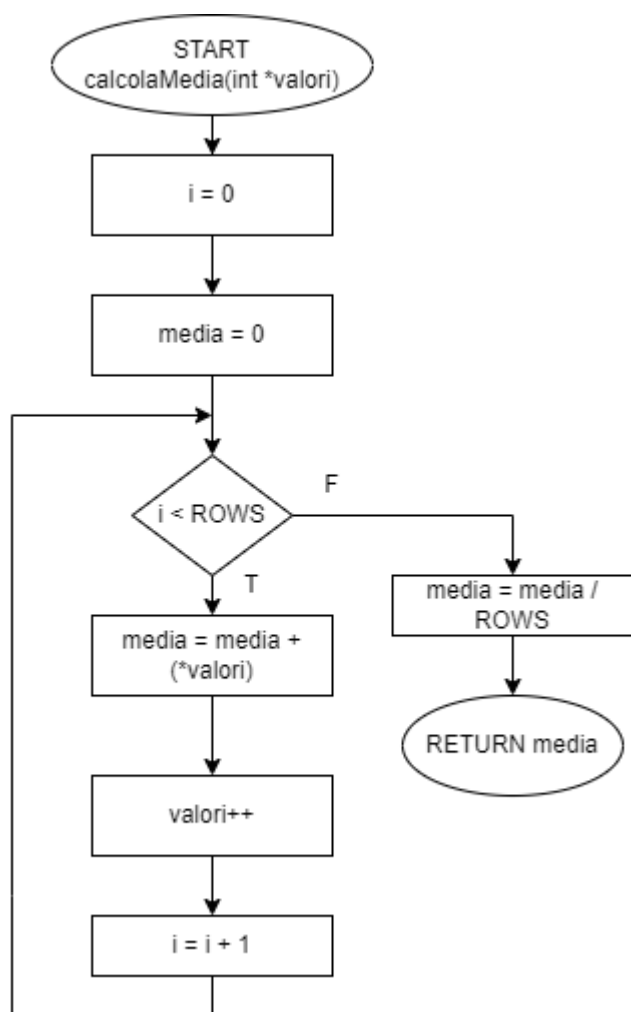


Funzione calcolaMedia()

La funzione calcola la media dei valori contenuti nella collezione e lo restituisce.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
valori	int*	Puntatore all'array di valori interi. Parametro formale	I	-
media	float	Media dei valori della collezione	L	-
i	int	Contatore del ciclo di caricamento	L	-

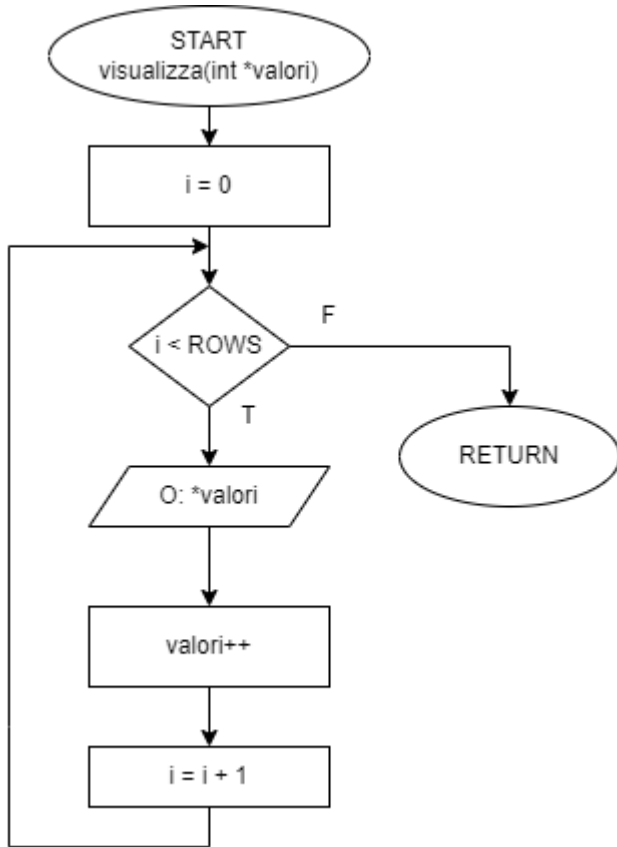
Flow-chart calcolaMedia()**Funzione visualizza()**

La funzione visualizza i valori contenuti nella collezione.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
valori	int*	Puntatore all'array di valori interi. Parametro formale	I	-
media	float	Media dei valori della collezione	L	-
i	int	Contatore del ciclo di caricamento	L	-

Flow-chart visualizza()



Codice con l'uso dei puntatori

```
/**
 * Caricamento di una collezione di interi, calcolo della
 * media e visualizzazione della collezione e della media
 * calcolata.
 * @author mgm
 */
#include <stdio.h>
//dimensione massima dell'array
#define ROWS 5

void carica(int*); // funzione di caricamento
float calcolaMedia(int*); // funzione di calcolo della media
void visualizza(int*); // funzione di visualizzazione

int main(void) {
    int numeri[ROWS]; // valori della collezione
    float media; // media dei valori della collezione

    carica(numeri);
    media = calcolaMedia(numeri);
    visualizza(numeri);
    printf("La media dei valori è %.2f\n", media);
}
```

```
    return 0;
}

/**
 * Caricamento dei valori della collezione
 * @param *valori puntatore all'array di valori
 */
void carica(int* valori) {
    for (int i = 0; i < ROWS; i++) {
        printf("Inserisci il valore %d all'indirizzo %p: ", i+1, valori);
        scanf("%d", valori++);
    }
}

/**
 * Calcola la media dei valori
 * @param *valori puntatore all'array di valori
 * @return la media dei valori
 */
float calcolaMedia(int* valori) {
    float media = 0;
    for (int i = 0; i < ROWS; i++) {
        media += *valori;
        valori++;
    }
    return media/ROWS;
}

/**
 * Visualizza i valori della collezione
 * @param *valori puntatore all'array di valori
 */
void visualizza(int* valori) {
    printf("Valori inseriti:\n");
    for (int i = 0; i < ROWS; i++) {
        printf("Valore %d all'indirizzo %p: %d\n", i+1, valori, *valori);
        valori++;
    }
}
```

Uso di array nelle funzioni per la gestione dell'array di int e di stringhe

Testo

Caricare una collezione di valori interi e una collezione di stringhe. Calcolare la media e visualizzare la collezione di interi con la media insieme alle stringhe.

Macroanalisi

L'applicazione ha lo scopo di testare una serie di funzioni che saranno utili per i futuri progetti. Questa applicazione prevede il caricamento di una collezione di stringhe e di interi e, successivamente la visualizzazione delle stringhe, dei valori interi e della media dei valori numerici.

Microanalisi

L'applicazione prevede l'uso di funzioni separate per la gestione dei valori delle collezioni, invocate all'interno del main(). La prima funzione prevede il caricamento delle stringhe e, successivamente viene invocata la funzione di caricamento dei valori numerici interi. A questo punto viene effettuato il calcolo della media invocando la funzione calcolaMedia() per passare subito dopo alla fase di visualizzazioni. Queste ultime sono gestite da funzioni separate, una per visualizzare la collezione delle stringhe e l'altra per visualizzare la collezione di valori interi. La visualizzazione della media viene invece direttamente gestita all'interno del main().

Tabella delle variabili e delle macro o costanti globali

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
ROWS	macro	Dimensione massima degli array	L	Inizializzata a 5
COLS	macro	Dimensione massima di una stringa	L	Inizializzato a 21

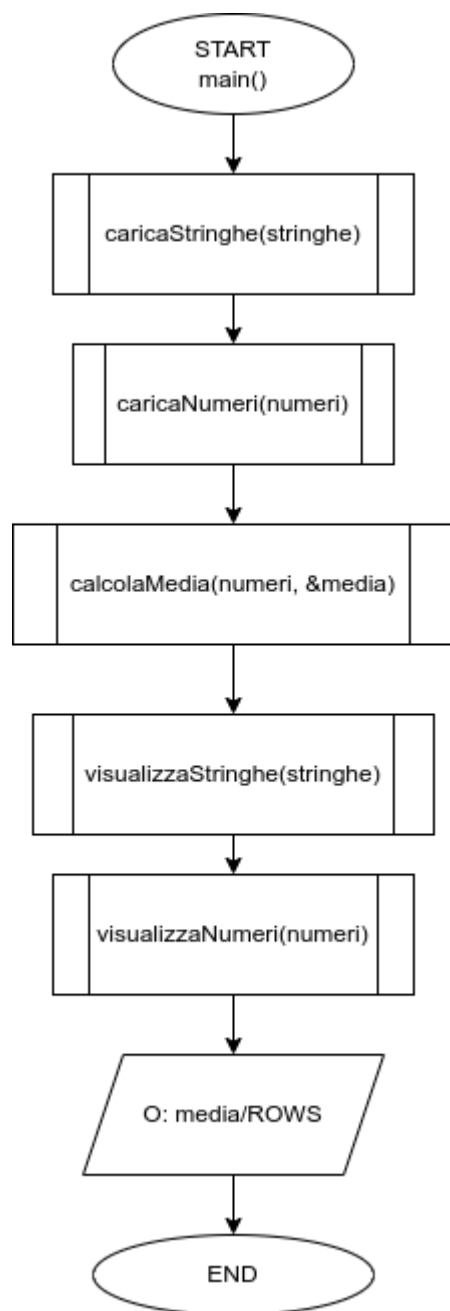
Funzione main()

La funzione main() serve per coordinare la sequenza logica di invocazione delle funzioni di caricamento, di calcolo della media e di visualizzazione.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
numeri[ROWS]	int	Array di valori interi	I/O	-
media	float	Media dei valori contenuti nell'array	O	-
stringhe[ROWS][COLS]	char	Array di stringhe	I/O	-

Flow-chart



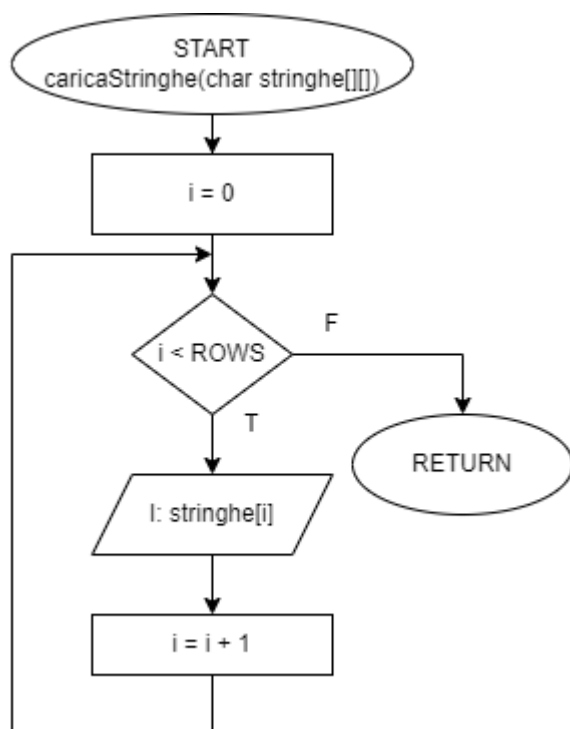
Funzione caricaStringhe()

La funzione effettua il caricamento delle stringhe memorizzandole nell'array passato come parametro.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
stringhe[ROWS][COLS]	char	Array di stringhe. Parametro formale	I	-
i	int	Contatore del ciclo di caricamento	L	-

Flow-chart caricaStringhe()

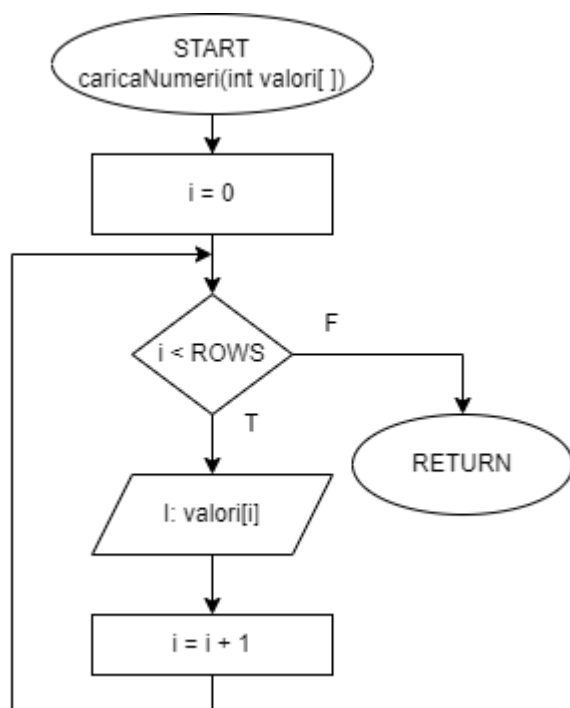


Funzione caricaNumeri()

La funzione effettua il caricamento dei valori della collezione memorizzandoli nell'array passato come parametro.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
valori[ROWS]	int	Array di valori interi. Parametro formale	I	-
i	int	Contatore del ciclo di caricamento	L	-

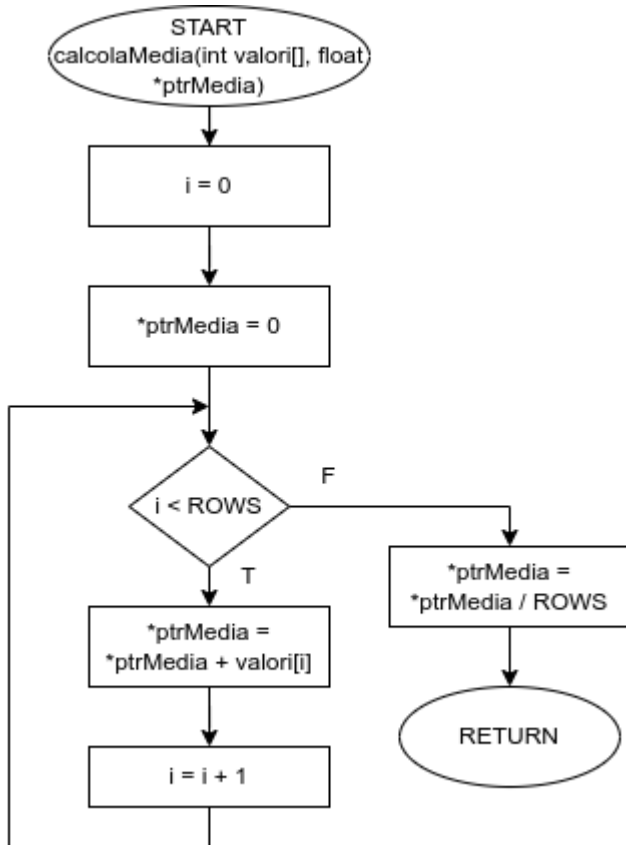
Flow-chart caricaNumeri()**Funzione calcolaMedia()**

La funzione calcola la media dei valori interi contenuti nella collezione e la salva nel parametro formale media, puntatore alla variabile presente nel main().

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
valori[ROWS]	int	Array di valori interi. Parametro formale	I	-
ptrMedia	float*	Puntatore alla variabile media nel main() usata per contenere la media dei valori della collezione. Parametro formale.	L	-
i	int	Contatore del ciclo di caricamento	L	-

Flow-chart calcolaMedia()



In questo caso ho deciso di passare l'indirizzo della variabile media dichiarata nella funzione main() come parametro attuale nel punto di chiamata della funzione: `calcolaMedia(numeri, &media)`. Optando per questa soluzione la funzione `calcolaMedia()` deve essere definita con due parametri formali, un array e un puntatore, quest'ultimo usato per agire direttamente sulla variabile media del main(). All'interno della funzione `calcolaMedia()` si utilizza l'operatore di dereferenziazione `*` per indicare che si vuole accedere al contenuto puntato dal parametro `ptrMedia`. La funzione non necessita di restituire un valore in quanto la media viene automaticamente salvata nella variabile media del main() grazie all'uso del puntatore `ptrMedia`.

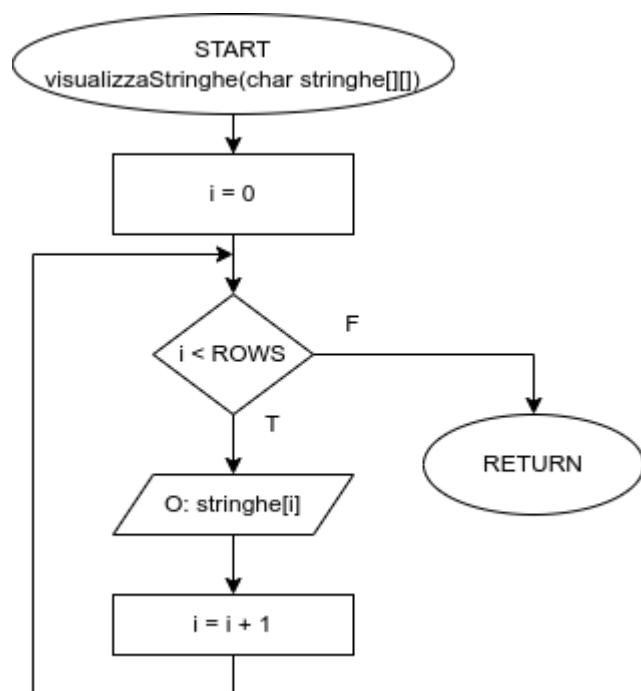
Funzione visualizzaStringhe()

La funzione visualizza le stringhe presenti nell'array passato come parametro.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
stringhe[ROWS] [COLS]	char	Array di stringhe. Parametro formale	O	-
i	int	Contatore del ciclo di visualizzazione	L	-

Flow-chart visualizzaStringhe()



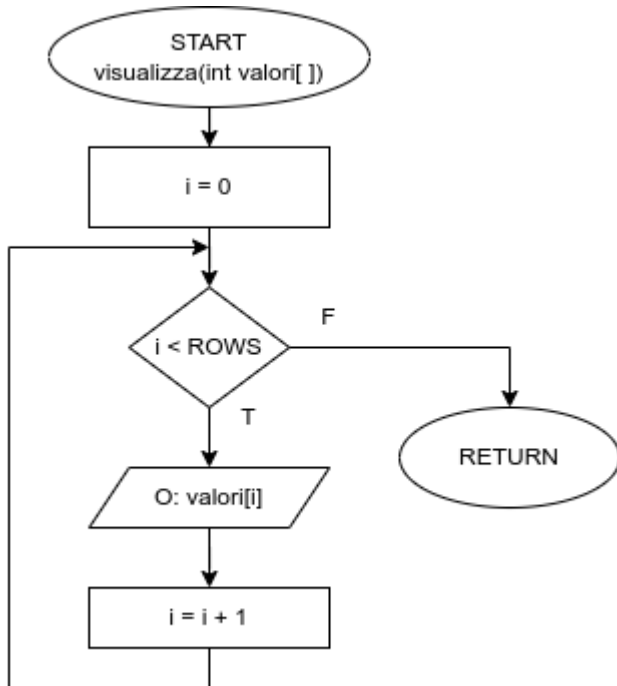
Funzione visualizzaNumeri()

La funzione visualizza i valori interi presenti nell'array passato come parametro.

Tabella delle variabili e delle macro o costanti

Nome	Tipo	Descrizione	I/O/L	Dominio di validità
valori[ROWS]	int	Array di interi. Parametro formale	O	-
i	int	Contatore del ciclo di visualizzazione	L	-

Flow-chart visualizzaNumeri()



Codice con l'uso dei di array e puntatore

```
/**
 * Caricamento di una collezione di interi, calcolo della
 * media e visualizzazione della collezione e della media
 * calcolata.
 * @author mgm
 */
#include <stdio.h>
//dimensione massima dell'array
#define ROWS 5
#define COLS 21

void caricaStringhe(char stringhe[ROWS][COLS]); // carica stringhe
void caricaNumeri(int valori[]); // carica numeri
void calcolaMedia(int valori[], float *ptrMedia); // calcola media
void visualizzaNumeri(int valori[]); // visualizza numeri
void visualizzaStringhe(char stringhe[][COLS]); // visualizza stringhe

int main(void) {
    char stringhe[ROWS][COLS]; // array di stringhe
    int numeri[ROWS]; // array di numeri interi
    float media; // media dei valori della collezione

    caricaStringhe(stringhe);
    caricaNumeri(numeri);
    calcolaMedia(numeri, &media);
    visualizzaStringhe(stringhe);
    visualizzaNumeri(numeri);
}
```

```
        printf("La media dei valori è %.2f\n", media/ROWS);
    return 0;
}

/**
 * Caricamento delle stringhe
 * @param stringhe[][] array di stringhe
 */
void caricaStringhe(char stringhe[ROWS][COLS]) {
    for (int i = 0; i < ROWS; i++) {
        printf("Inserire la %d^ stringa: ", i+1);
        scanf("%s", stringhe[i]);
    }
}

/**
 * Caricamento dei valori numerici interi
 * @param *valori puntatore all'array di valori
 */
void caricaNumeri(int valori[]) {
    for (int i = 0; i < ROWS; i++) {
        printf("Inserisci il valore %d: ", i+1);
        scanf("%d", &valori[i]);
    }
}

/**
 * Calcola la media dei valori
 * @param *valori puntatore all'array di valori
 * @return la media dei valori
 */
void calcolaMedia(int valori[], float *ptrMedia) {
    *ptrMedia = 0;
    for (int i = 0; i < ROWS; i++) {
        *ptrMedia += valori[i];
    }
}

/**
 * Visualizza l'array di stringhe
 * @param stringhe[][] array di stringhe
 */
void visualizzaStringhe(char stringhe[][COLS]) {
    printf("Stringhe inserite:\n");
    for (int i = 0; i < ROWS; i++) {
        printf("Valore %d all'indirizzo %p: %s\n", i+1, &stringhe[i],
stringhe[i]);
    }
}
```

```
/**
 * Visualizza i valori interi nell'array
 * @param valori[] array di valori interi
 */
void visualizzaNumeri(int valori[]) {
    printf("Valori inseriti:\n");
    for (int i = 0; i < ROWS; i++) {
        printf("Valore  %d  all'indirizzo  %p:  %d\n",  i+1,  &valori[i],
valori[i]);
    }
}
```

D - OLD Esempio d'uso dei file

Soluzione usando i file di testo

```
/**
 * Progettare un algoritmo che permetta di caricare e salvare su file i dati dei
 * corsi universitari tenuti da diversi docenti, con il relativo voto medio
 * degli esami.
 * L'algoritmo deve permettere di visualizzare:
 * a) l'elenco dei corsi con il nome del docente e il voto medio degli esami in
 * ordine crescente di voto medio;
 * b) visualizzare il corso e il voto medio di un docente.
 * Si facciano tutte le ipotesi aggiuntive sulle strutture dati da utilizzare e
 * si scelga come algoritmo di ordinamento e di ricerca uno fra quelli studiati
 * a lezione.
 * Come algoritmo di ricerca si usi l'algoritmo di ricerca binario.
 * Si usino le funzioni per gestire tutti i distinti aspetti logici
 * dell'algoritmo in un'ottica di progettazione top-down.
 * https://cboard.cprogramming.com/c-programming/150023-passing-file-pointer-function.html
 * @author mgm
 * @version 2.0
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define ARCHIVIO "corsi.txt"
#define DIM 10

typedef struct {
    char nomeCorso[20];
    char docente[50];
    float votoMedio;
} tipo_corso;

// Prototipi funzioni
void menu(int *scelta); // gestione menù
void caricaDati(FILE *filePointer); // caricamento corsi, docenti e voti
void elencoPerVotoMedio(FILE *filePointer); // elenco di corso, docente e voto in
ordine crescente di voto
void datiDiDocente(); // corso e voto di un docente inserito in input
int corsiInArray(FILE *filePointer, tipo_corso corsi[]);
void ordinaPerVoto(tipo_corso corsi[], int nCorsi);
void datiDiDocente(FILE *filePointer); // visualizza corso e voto di un docente
void visualizzaCorsi(tipo_corso corsi[], int nCorsi, int posizione); // visualizza
un solo elemento o l'elenco completo a seconda del valore del parametro formale
posizione
void ordinaPerDocente(tipo_corso corsi[], int nCorsi); // ordinamento dell'elenco
```

per docente

```
int ricercaDocente(tipo_corso corsi[], int nCorsi, char docente[]); // ricerca del  
docente nell'elenco tramite una ricerca binaria
```

```
int main()  
{  
    tipo_corso corsi[DIM];  
    int scelta;  
  
    FILE *filePointer;  
  
    filePointer = fopen(ARCHIVIO, "a+");  
    if(filePointer == NULL) {  
        printf("Errore nell'apertura del file\n");  
        exit(1);  
    }  
  
    do {  
        /*START debug*/  
        //int nCorsi = corsiInArray(filePointer, corsi);  
        //visualizzaCorsi(corsi, nCorsi, 0);  
        /*END debug*/  
        // Visualizza il menù e permette di far scegliere cosa fare  
        menu(&scelta);  
        // Esegue ciò che è stato scelto  
        switch (scelta) {  
            case 1:  
                // Caricamento dati nel file  
                caricaDati(filePointer);  
                break;  
            case 2:  
                // Elenco in ordine crescente di voto medio  
                elencoPerVotoMedio(filePointer);  
                break;  
            case 3:  
                // Visualizzazione dati di un docente  
                datiDiDocente(filePointer);  
                break;  
            case 4:  
                // Chiusura programma  
                printf("Chiusura del programma\n");  
                break;  
        }  
    } while (scelta != 4);  
  
    fclose(filePointer);  
    return 0;  
}
```

```
/**
 * Visualizza il menù e richiede l'inserimento della scelta.
 * @param scelta scelta del menù
 */
void menu(int *scelta) {
    printf("1) Caricamento dati\n");
    printf("2) Elenco corsi in ordine crescente di voto medio\n");
    printf("3) Dati di un docente\n");
    printf("4) Esci\n");
    printf("Scelta (1-4): ");
    scanf("%d", scelta);
    while(*scelta < 1 || *scelta > 4) {
        printf("Scelta (1-4): ");
        scanf("%d", scelta);
    }
}

/**
 * Caricamento dei dati nel file "corsi.txt"
 * @param filePointer puntatore al file
 */
void caricaDati(FILE *filePointer) {
    tipo_corso corso; // singolo corso da caricare

    // Svuotamento dello standard input
    while(getchar() != '\n');
    // Input dei dati
    printf("Corso: ");
    fgets(corso.nomeCorso, sizeof(corso.nomeCorso), stdin);
    corso.nomeCorso[strlen(corso.nomeCorso)-1] = '\0';

    printf("Nominativo del docente: ");
    fgets(corso.docente, sizeof(corso.docente), stdin);
    corso.docente[strlen(corso.docente)-1] = '\0';

    printf("Voto medio: ");
    scanf("%f", &corso.votoMedio);

    // Per scrivere nel file di testo il voto medio devo convertirlo in una stringa,
    // e per farlo uso la funzione sprintf(), usando come stringa destinataria
    // una stringa di almeno 6 caratteri.
    // https://www.geeksforgeeks.org/what-is-the-best-way-in-c-to-convert-a-number-to-a-string/
    char strVoto[6];
    sprintf(strVoto, "%.2f", corso.votoMedio);
    // Scrittura su file
    fprintf(filePointer, "%s,%s,%s\n", corso.nomeCorso, corso.docente, strVoto);
}

/**
 * Caricamento dei corsi in un array.
```

```
* @param filePointer puntatore al file
* @param corsi array in cui caricare i corsi del file
*/
int corsiInArray(FILE *filePointer, tipo_corso corsi[]) {
    int i;
    char strVoto[7];
    char stringa[80];

    // Riporto il puntatore del file all'inizio
    rewind(filePointer);
    for(i = 0; !feof(filePointer); i++) {
        // Si procede al caricamento dell'array solo se non si è raggiunta la fine
        // del file
        if(fgets(stringa, 80, filePointer) != NULL) {
            //printf("%s", stringa); /**per debug
            // Per recuperare ogni singola parte della riga letta dal file si usa la funzione
            // strtok() che divide una stringa usando un delimitatore. La prima invocazione
            // do strtok() fa riferimento alla riga letta dal file (stringa), le successive
            // usano NULL perché in questo modo strtok() parte con il nuovo token dalla
            // posizione in cui è arrivata.
            // https://www.codingame.com/playgrounds/14213/how-to-play-with-strings-in-c/string-split
            strcpy(corsi[i].nomeCorso, strtok(stringa, ","));
            strcpy(corsi[i].docente, strtok(NULL, ","));
            strcpy(strVoto, strtok(NULL, "\n"));
            // Per il voto medio si usa la stringa strVoto, e si converte
            // in un float usando la funzione di libreria atof() di stdlib.h
            // http://www.cplusplus.com/reference/cstdlib/atof/
            corsi[i].votoMedio = atof(strVoto);
        }
    }

    return(i-1);
}

/**
 * Visualizzazione dell'elenco dei corsi in ordine crescente di voto medio.
 * @param filePointer puntatore al file
 */
void elencoPerVotoMedio(FILE *filePointer) {
    tipo_corso corsi[DIM]; // elenco di corsi caricati su file
    int nCorsi; // numero corsi caricati

    nCorsi = corsiInArray(filePointer, corsi);
    ordinaPerVoto(corsi, nCorsi);
    visualizzaCorsi(corsi, nCorsi, 0);
}

/**
 * Ordinamento dell'array di corsi in ordine crescente di voto medio.
 * @param corsi array di corsi da ordinare
```

```
* @param nCorsi  numero di corsi caricati
*/
void ordinaPerVoto(tipo_corso corsi[], int nCorsi) {
    tipo_corso temp;

    for (int i = 1; i < nCorsi; i++) {
        for (int j = 0; j < (nCorsi-i); j++) {
            if (corsi[j].votoMedio > corsi[j+1].votoMedio) {
                temp = corsi[j];
                corsi[j] = corsi[j+1];
                corsi[j+1] = temp;
            }
        }
    }
}

/**
 * Visualizzazione dell'elenco dei corsi o di un corso specifico.
 * @param corsi  array di corsi
 * @param nCorsi  numero di corsi caricati
 * @param posizione = 0 - visualizza tutto l'elenco
 *                  > 0 - visualizza i dati della posizione indicata
 */
void visualizzaCorsi(tipo_corso corsi[], int nCorsi, int posizione) {
    if (posizione == 0) {
        for (int i = 0; i < nCorsi; i++) {
            printf("%s, %s, %f\n", corsi[i].nomeCorso, corsi[i].docente,
corsi[i].votoMedio);
        }
    } else {
        printf("%s, %s, %f\n", corsi[posizione].nomeCorso, corsi[posizione].docente,
corsi[posizione].votoMedio);
    }
}

/**
 * Visualizzazione dei dati del corso e del voto medio di un docente
 * inserito in input.
 */
void datiDiDocente(FILE *filePointer) {
    tipo_corso corsi[DIM];    // elenco di corsi caricati su file
    int nCorsi;               // numero corsi caricati
    int posizione;            // posizione del docente
    char docente[50];

    nCorsi = corsiInArray(filePointer, corsi);
    ordinaPerDocente(corsi, nCorsi);
    while(getchar() != '\n');
    printf("Inserire il nome del docente: ");
    fgets(docente, sizeof(docente), stdin);
}
```

```
    docente[strlen(docente)-1] = '\\0';
    posizione = ricercaDocente(corsi, nCorsi, docente);
    if (posizione == -1) {
        printf("Docente inesistente\\n");
    } else {
        visualizzaCorsi(corsi, nCorsi, posizione);
    }
}

/**
 * Ordinamento dell'array di corsi in ordine crescente di docente.
 * @param corsi array di corsi da ordinare
 * @param nCorsi numero di corsi caricati
 */
void ordinaPerDocente(tipo_corso corsi[], int nCorsi) {
    tipo_corso temp;
    //bool terminare = true;

    for (int i = 1; i < nCorsi; i++) {
        for (int j = 0; j < (nCorsi-i); j++) {
            if (strcmp(corsi[j].docente, corsi[j+1].docente) > 1) {
                temp = corsi[j];
                corsi[j] = corsi[j+1];
                corsi[j+1] = temp;
            }
        }
    }
}

/**
 * Ricerca binaria di un docente nell'array dei corsi.
 * @param corsi array di corsi in cui cercare
 * @param nCorsi numero di corsi caricati
 * @param docente nome del docente da cercare nell'array dei corsi
 */
int ricercaDocente(tipo_corso corsi[], int nCorsi, char docente[]) {
    int low = 0;
    int high = nCorsi - 1;
    int mid = 0;

    while (low <= high) {
        mid = low + (high - low) / 2;
        if (strcmp(corsi[mid].docente, docente) == 0) {
            return mid;
        }
        if (strcmp(corsi[mid].docente, docente) < 0) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
}
```

```
    }  
    return -1;  
}
```

Soluzione usando i file binari

```
/**  
 * Progettare un algoritmo che permetta di caricare e salvare su file i dati dei  
 * corsi universitari tenuti da diversi docenti, con il relativo voto medio  
 * degli esami.  
 * L'algoritmo deve permettere di visualizzare:  
 * a) l'elenco dei corsi con il nome del docente e il voto medio degli esami in  
 * ordine crescente di voto medio;  
 * b) visualizzare il corso e il voto medio di un docente.  
 * Si facciano tutte le ipotesi aggiuntive sulle strutture dati da utilizzare e  
 * si scelga come algoritmo di ordinamento e di ricerca uno fra quelli studiati  
 * a lezione.  
 * Come algoritmo di ricerca si usi l'algoritmo di ricerca binario.  
 * Si usino le funzioni per gestire tutti i distinti aspetti logici  
 * dell'algoritmo in un'ottica di progettazione top-down.  
 * https://cboard.cprogramming.com/c-programming/150023-passing-file-pointer-function.html  
 * @author mgm  
 * @version 1.0  
 */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdbool.h>  
  
#define ARCHIVIO "corsi.bin"  
#define DIM 10  
  
typedef struct {  
    char nomeCorso[20];  
    char docente[50];  
    float votoMedio;  
} tipo_corso;  
  
// Prototipi funzioni  
void menu(int *scelta); // gestione menù  
void caricaDati(); // caricamento corsi, docenti e voti  
void elencoPerVotoMedio(); // elenco di corso, docente e voto in ordine crescente di  
voto  
void datiDiDocente(); // corso e voto di un docente inserito in input  
int corsiInArray(tipo_corso corsi[]);  
void ordinaPerVoto(tipo_corso corsi[], int nCorsi);  
void datiDiDocente(); // visualizza corso e voto di un docente  
void visualizzaCorsi(tipo_corso corsi[], int nCorsi, int posizione); // visualizza  
un solo elemento o l'elenco completo a seconda del valore del parametro formale  
posizione
```

```
void ordinaPerDocente(tipo_corso corsi[], int nCorsi); // ordinamento dell'elenco
per docente
int ricercaDocente(tipo_corso corsi[], int nCorsi, char docente[]); // ricerca del
docente nell'elenco tramite una ricerca binaria

int main()
{
    tipo_corso corsi[DIM];
    int scelta;

    do {
        /*START debug*/
        //int nCorsi = corsiInArray(corsi);
        //visualizzaCorsi(corsi, nCorsi, 0);
        /*END debug*/
        // Visualizza il menù e permette di far scegliere cosa fare
        menu(&scelta);
        // Esegue ciò che è stato scelto
        switch (scelta) {
            case 1:
                // Caricamento dati nel file
                caricaDati(/*filePointer*/);
                break;
            case 2:
                // Elenco in ordine crescente di voto medio
                elencoPerVotoMedio();
                break;
            case 3:
                // Visualizzazione dati di un docente
                datiDiDocente();
                break;
            case 4:
                // Chiusura programma
                printf("Chiusura del programma\n");
                break;
        }

    } while (scelta != 4);

    return 0;
}

/**
 * Visualizza il menù e richiede l'inserimento della scelta.
 * @param scelta scelta del menù
 */
void menu(int *scelta) {
    printf("1) Caricamento dati\n");
    printf("2) Elenco corsi in ordine crescente di voto medio\n");
    printf("3) Dati di un docente\n");
}
```

```
printf("4) Esci\n");
printf("Scelta (1-4): ");
scanf("%d", scelta);
while(*scelta < 1 || *scelta > 4) {
    printf("Scelta (1-4): ");
    scanf("%d", scelta);
}
}

/**
 * Caricamento dei dati nel file "corsi.bin"
 */
void caricaDati() {
    tipo_corso corso; // singolo corso da caricare

    FILE *filePointer;

    filePointer = fopen(ARCHIVIO, "ab");
    if(filePointer == NULL) {
        printf("Errore nell'apertura del file\n");
        exit(1);
    }
    // Svuotamento dello standard input
    while(getchar() != '\n');
    // Input dei dati
    printf("Corso: ");
    fgets(corso.nomeCorso, sizeof(corso.nomeCorso), stdin);
    corso.nomeCorso[strlen(corso.nomeCorso)-1] = '\0';

    printf("Nominativo del docente: ");
    fgets(corso.docente, sizeof(corso.docente), stdin);
    corso.docente[strlen(corso.docente)-1] = '\0';

    printf("Voto medio: ");
    scanf("%f", &corso.votoMedio);

    // Scrittura su file
    fwrite(&corso, sizeof(tipo_corso), 1, filePointer);

    fclose(filePointer);
}

/**
 * Caricamento dei corsi in un array.
 * @param corsi array in cui caricare i corsi del file
 */
int corsiInArray(tipo_corso corsi[]) {
    int i;

    FILE *filePointer;
```

```
filePointer = fopen(ARCHIVIO, "rb");
if(filePointer == NULL) {
    printf("Errore nell'apertura del file\n");
    exit(1);
}

for(i = 0; !feof(filePointer); i++) {
    fread(&corsi[i], sizeof(tipo_corso), 1, filePointer);
}

fclose(filePointer);
return(i-1);
}

/**
 * Visualizzazione dell'elenco dei corsi in ordine crescente di voto medio.
 */
void elencoPerVotoMedio() {
    tipo_corso corsi[DIM];    // elenco di corsi caricati su file
    int nCorsi;               // numero corsi caricati

    nCorsi = corsiInArray(corsi);
    ordinaPerVoto(corsi, nCorsi);
    visualizzaCorsi(corsi, nCorsi, 0);
}

/**
 * Ordinamento dell'array di corsi in ordine crescente di voto medio.
 * @param corsi array di corsi da ordinare
 * @param nCorsi numero di corsi caricati
 */
void ordinaPerVoto(tipo_corso corsi[], int nCorsi) {
    tipo_corso temp;

    for (int i = 1; i < nCorsi; i++) {
        for (int j = 0; j < (nCorsi-i); j++) {
            if (corsi[j].votoMedio > corsi[j+1].votoMedio) {
                temp = corsi[j];
                corsi[j] = corsi[j+1];
                corsi[j+1] = temp;
            }
        }
    }
}

/**
 * Visualizzazione dell'elenco dei corsi o di un corso specifico.
 * @param corsi array di corsi
 * @param nCorsi numero di corsi caricati
```

```
* @param posizione = 0 - visualizza tutto l'elenco
*                  > 0 - visualizza i dati della posizione indicata
*/
void visualizzaCorsi(tipo_corso corsi[], int nCorsi, int posizione) {
    if (posizione == 0) {
        for (int i = 0; i < nCorsi; i++) {
            printf("%s, %s, %f\n", corsi[i].nomeCorso, corsi[i].docente,
corsi[i].votoMedio);
        }
    } else {
        printf("%s, %s, %f\n", corsi[posizione].nomeCorso, corsi[posizione].docente,
corsi[posizione].votoMedio);
    }
}

/**
 * Visualizzazione dei dati del corso e del voto medio di un docente
 * inserito in input.
 */
void datiDiDocente() {
    tipo_corso corsi[DIM];    // elenco di corsi caricati su file
    int nCorsi;               // numero corsi caricati
    int posizione;            // posizione del docente
    char docente[50];

    nCorsi = corsiInArray(corsi);
    ordinaPerDocente(corsi, nCorsi);
    while(getchar() != '\n');
    printf("Inserire il nome del docente: ");
    fgets(docente, sizeof(docente), stdin);
    docente[strlen(docente)-1] = '\0';
    posizione = ricercaDocente(corsi, nCorsi, docente);
    if (posizione == -1) {
        printf("Docente inesistente\n");
    } else {
        visualizzaCorsi(corsi, nCorsi, posizione);
    }
}

/**
 * Ordinamento dell'array di corsi in ordine crescente di docente.
 * @param corsi array di corsi da ordinare
 * @param nCorsi numero di corsi caricati
 */
void ordinaPerDocente(tipo_corso corsi[], int nCorsi) {
    tipo_corso temp;
    //bool terminare = true;

    for (int i = 1; i < nCorsi; i++) {
        for (int j = 0; j < (nCorsi-i); j++) {
```

```
        if (strcmp(corsi[j].docente, corsi[j+1].docente) > 1) {
            temp = corsi[j];
            corsi[j] = corsi[j+1];
            corsi[j+1] = temp;
        }
    }
}

/**
 * Ricerca binaria di un docente nell'array dei corsi.
 * @param corsi array di corsi in cui cercare
 * @param nCorsi numero di corsi caricati
 * @param docente nome del docente da cercare nell'array dei corsi
 */
int ricercaDocente(tipo_corso corsi[], int nCorsi, char docente[]) {
    int low = 0;
    int high = nCorsi - 1;
    int mid = 0;

    while (low <= high) {
        mid = low + (high - low) / 2;
        if (strcmp(corsi[mid].docente, docente) == 0) {
            return mid;
        }
        if (strcmp(corsi[mid].docente, docente) < 0) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
```

Bibliografia e sitografia

[Manoj Jaggarapu](#), [Design Principles and Patterns](#), ultima visita marzo 2022

[Claudio Fornaro](#), [Corso di programmazione in C \(Standard C89/C90\)](#), ultima accesso novembre 2022