POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Algoritmi e Strutture Dati: Introduzione

Gianpiero Cabodi e Paolo Camurati

Obiettivi

Corso fondante del CdL Ing. Informatica, prosegue e approfondisce Informatica e soprattutto Tecniche di Programmazione

Algoritmi e strutture dati:
- analisi di complessità
- strutture dati e algoritmi fondamentali

Programmazione avanzata in C:
- ricorsione
- puntatori e strutture dinamiche
- modularità

} Problem-solving avanzato

Problem-solving **É un'attività creativa**

Abilità richieste:

- sapere (concetti, modelli, linguaggio, etc)
- saper fare (programmazione)
- progettare (formalizzazione, scelte)

Difficoltà:

- oggettiva: è un'attività creativa e progettuale
- soggettiva:
  - **disuniformità** di preparazione pregressa
  - **disuniformità** di capacità/applicazione
  - tempi **diversi** di apprendimento

Metodo di studio/lavoro

Il metodo è importante quanto il contenuto: in generale in assenza di metodo:

- allo sforzo fatto non corrisponde un risultato adeguato
- o non corrisponde affatto un risultato.

Il metodo si articola su 3 pilastri:

1. sapere
2. saper fare/eseguire (programmazione)
3. progettare (problem-solving).

1. sapere (concetti, modelli, linguaggio, etc)
   - tipo di studio/lavoro: in prevalenza studiare, capire, ricordare
   - attività: seguire lezioni, leggere materiale proposto identificando i concetti chiave
2. saper fare/eseguire (programmazione)
   - prerequisiti: studiare (1. sapere) linguaggio C, algoritmi e tecniche insegnati a lezione
   - tipo di lavoro: applicare/eseguire, attenzione ai dettagli, precisione, ordine, metodo
   - attività: capire/analizzare programmi già scritti, realizzare funzioni/programmi con algoritmi e strutture dati già decise (es. funzione con prototipo e algoritmo dati), correggere errori usando il debugger

3. progettare (problem solving)
   - prerequisiti: 1. sapere, 2. saper fare/eseguire
   - tipo di lavoro:
     - capire e risolvere problemi (comprensione del testo, capacità di analisi)
     - intuire, creare (logica, modelli astratti, ragionamento deduttivo e induttivo)
     - scomporre in sotto-problemi, riutilizzare e modificare tecniche e/o soluzioni apprese (metodo, analisi, astrazione)
   - attività: esercizi spiegati a lezione, laboratori, esami proposti
     - soluzioni individuali, confronto con soluzioni esistenti, metodo incrementale (un pezzo alla volta)

Non basta studiare la teoria, non basta fare solo programmazione, bisongia fare entrambi

Sapere → Saper fare → Saper fare bene

SAPERE: non basta ricordare superficialmente qualche concetto (solo memoria e basta), occorre **CAPIRLI** per :

- **applicarli** (ai problemi di programmazione, agli esercizi dello scritto di teoria)
- **esprimerli CORRETTAMENTE** (all'esame orale)

DAL SAPERE AL SAPER FARE: per risolvere i problemi di programmazione:

- atteggiamento errato: illudersi di risolvere un problema come se si ponesse per la prima volta, senza conoscere quanto già scoperto
- atteggiamento corretto: conoscere e capire la teoria sottostante per poi applicarla.

SAPER FARE BENE: non basta che il programma «funzioni»:

- deve essere efficiente, leggibile, manutenibile, affidabile etc etc

Prerequisiti

Informatica: basi di programmazione

Tecniche di Programmazione

- linguaggio C (compreso il concetto di puntatore) **forse**
- problem solving elementare **forse**
- analisi di complessità

Matematica

- algebra di Boole
- logica
- insiemistica
- calcolo combinatorio

Comprensione di testi complessi e formulazione di problemi

- astrazione
- ragionamento deduttivo e induttivo

Obiettivi in dettaglio

Programmi modulari:

- scomposizione in sotto-problemi
- uso di funzioni
- strutture dati modulari
- in pratica: norme di buona progettazione *(«non fare i praticoni»)*

Puntatori e allocazione dinamica:

- C avanzato (puntatori)
- strutture dati dinamiche per collezioni di dati (ADT)

programmazione a oggetti manuale

Algoritmi ricorsivi:

- estendono i limiti di costrutti condizionali e iterativi
- difficili perché richiedono formulazione e progettazione di tipo *«divide et impera»*
- presentati con modelli dal calcolo combinatorio

Algoritmi classici: come sono stati risolti in letteratura (in modo efficiente) problemi su:

- collezioni di dati e code, tabelle di simboli, alberi e grafi

A cosa serve il corso?

- Corsi successivi (L, LM)
- Tirocini
- Interview per l'assunzione
- Lavoro
- Ambiti di uso degli algoritmi e delle strutture dati: tutta l'Informatica/Data Science
- Ambiti di uso del C:
  - Sistemi Operativi
  - Sistemi Embedded
  - Internet of Things

Esempio: preparare Technical Interview di Google

Preparing for Google Technical Internship Interviews

https://www.youtube.com/watch?v=ko-KkSmp-Lk#action=share

Preparing for your Interview

Technical Preparation

Coding: Google Engineers primarily code in C++, Java, or Python. We ask that you use one of these languages during your interview. For phone interviews, you will be asked to write code real time in Google Docs. You may be asked to:

- Construct / traverse data structures
- Implement system routines
- Distill large data sets to single values
- Transform one data set to another

Algorithms: You will be expected to know the complexity of an algorithm and how you can improve/change it. You can find examples that will help you prepare on TopCoder. Some examples of algorithmic challenges you may be asked about include:

- Big-O analysis: understanding this is particularly important
- Sorting and hashing
- Handling obscenely large amounts of data

Sorting: We recommend that you know the details of at least one n*log(n) sorting algorithm, preferably two (say, quicksort and merge sort). Merge sort can be highly useful in situations where quicksort is impractical, so take a look at it. What common sorting functions are there? On what kind of input data are they efficient, when are they not? What does efficiency mean in these cases in terms of runtime and space used? E.g. in exceptional cases insertion-sort or radix-sort are much better than the generic QuickSort / MergeSort / HeapSort answers.

Reminder:
Engineering Practicum interviews will focus on coding, algorithms and data structures and content will be level appropriate. See slide 2.

Preparing for your Interview

Technical Preparation

Data structures: Study up on as many other structures and algorithms as possible. We recommend you know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem. Be able to recognize them when an interviewer asks you in disguise. Find out what NP-complete means. You will also need to know about Trees, basic tree construction, traversal and manipulation algorithms, hash tables, stacks, arrays, linked lists, priority queues.

Hashtables and Maps: Hashtables are arguably the single most important data structure known to mankind. You should be able to implement one using only arrays in your favorite language, in about the space of one interview. You'll want to know the O() characteristics of the standard library implementation for Hashtables and Maps in the language you choose to write in.

Trees: We recommend you know about basic tree construction, traversal and manipulation algorithms. You should be familiar with binary trees, n-ary trees, and trie-trees at the very least. You should be familiar with at least one flavor of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree. You'll want to know how it's implemented. You should know about tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

Min/Max Heaps: Heaps are incredibly useful. Understand their application and O() characteristics. We probably won't ask you to implement one during an interview, but you should know when it makes sense to use one.

Reminder:
Engineering Practicum interviews will focus on coding, algorithms and data structures and content will be level appropriate. See slide 2.

Preparing for your Interview

Technical Preparation

Graphs: To consider a problem as a graph is often a very good abstraction to apply, since well known graph algorithms for distance, search, connectivity, cycle-detection etc. will then yield a solution to the original problem. There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros/cons. You should know the basic graph traversal algorithms, breadth-first search and depth-first search. Know their computational complexity, their tradeoffs and how to implement them in real code.

Recursion: Many coding problems involve thinking recursively and potentially coding a recursive solution. Prepare for recursion, which can sometimes be tricky if not approached properly. Practice some problems that can be solved iteratively, but a more elegant solution is recursion.

Operating systems: You should understand processes, threads, concurrency issues, locks, mutexes, semaphores, monitors and how they all work. Understand deadlock, livelock and how to avoid them. Know what resources a process needs and a thread needs. Understand how context switching works, how it's initiated by the operating system and underlying hardware. Know a little about scheduling. The world is rapidly moving towards multi-core, so know the fundamentals of "modern" concurrency constructs.

Mathematics: Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because counting problems, probability problems and other Discrete Math 101 situations surrounds us. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of elementary probability theory and combinatorics. You should be familiar with n-choose-k problems and their ilk – the more the better.

Still want more info?
Tech interviews @ Google

Distributed systems & parallel programming

Scalable Web Architecture & Distributed systems

How search works

Statistiche superi Algoritmi e strutture dati

	2022/23	
Nuovi frequentanti	420	
	>= 1 volta	mai
prenotati	224	196
prenotati && relazione	215	9
superi <30/09	177 (42,1%)	

Tasso di supero rispetto a chi ha consegnato almeno una volta la relazione:

- 2022/23: 177/215 = 82%

Cosa ci dicono le statistiche?

Perché il 46,7% dei nuovi frequentanti dell'A.A. 2022/23 non si è mai prenotato nemmeno una volta?

- Lato corso:
  - difficile e impegnativo
  - la diminuazione a 8 cfu ne ha diminuito il peso relativo nel semestre
- Lato studente:
  - approccio greedy: mi concentro sul corso che vale di più come crediti e che è più probabile che io passi
  - il paradigma greedy non garantisce l'ottimalità della soluzione….

Cosa ci dicono le statistiche?

Perché il 57,9% dei nuovi frequentanti dell'A.A. 2022/23 non ha superato ancora l'esame?

- Lato corso: difficile e impegnativo
- Lato studente:
  - cattiva o mancata pianificazione delle proprie attività ("carico tutto quanto possibile e poi non seguo", "sono sotto la soglia di crediti e non do la precedenza agli esami in arretrato", etc)
  - a metà novembre lascio perdere tutto invece di impegnarmi di più
  - mancata frequenza di lezioni e laboratori

Cosa ci dice Tecniche di Programmazione?

- premessa: Tecniche di Programmazione (6 crediti) inserito al primo anno per
  - «spostare» un po' di carico dal secondo anno al primo
  - alleggerire «Algoritmi e Strutture Dati» (8 crediti) rispetto ad «Algoritmi e Programmazione» (12 crediti)
- carico percepito ELEVATO: secondo semestre impegnativo
- circa il 20% degli/delle studenti hanno Tecniche di Programmazione in carico ma NON lo frequentano e NON tentano l'esame.

- Esame (3 appelli)
  - superato dal 64% degli iscritti (80% se si escludono gli «inattivi»)
  - voti: >= 27 (40% degli/delle studenti) - <=23 (30%)
    Esame NON percepito come difficile in media (scelta consapevole dei docenti, voluta per tenere conto del carico complessivo e del carattere introduttivo del corso)
  - valutazione qualitativa: si notano forti differenze tra studenti: già programmato in precedenza – mai programmato, assimilati bene i contenuti del corso – preparazione decisamente fai-da-te
- ATTENZIONE: aver superato Tecniche di Programmazione con voto basso potrebbe indicare prerequisiti insufficienti per Algoritmi e Strutture Dati!

Trasferimenti al II anno

- Non hanno Tecniche di Programmazione:
  - chi non lo ha potuto inserire in carico al I anno
  - Chi si trasferisce da altri CdL.
- I contenuti di Tecniche di Programmazione sono dati per acquisiti
- Il materiale di Tecniche di Programmazione del 2022/23 è disponibile su Dropbox di ASD per coloro che frequenteranno il corso nel II semestre
- Esperienza pregressa:
  - alcuni preparano in simultanea Algoritmi e Tecniche
  - pochi superano Algoritmi prima o contemporaneamente al supero di Tecniche
  - la maggioranza supera prima Tecniche e poi Algoritmi.

Cosa serve per Algoritmi e Strutture Dati?

- prerequisiti (già citati)
- capacità logiche: comprendere il problema, identificare una strategia
- capacità pratiche: trasformare la strategia risolutiva in un programma efficiente e funzionante
- metodo efficace di lavoro (un misto tra studio, applicazione di tecniche apprese, problem solving)
- impegno personale continuo sul semestre:
  - seguire le lezioni per capire
  - programmare implementando i concetti capiti (richiede tempo!)
  - frequentare i laboratori e svolgere gli esercizi
  - non rinunciare a metà corso, dicendo «lo lascio come ultimo esame»
  - non «svegliarsi» 2 settimane prima dell'appello e stupirsi se non lo si passa!

- **NO**: studio solo teorico senza provare a programmare
- **NO**: limitarsi alla comprensione di programmi altrui
- **NO**: approccio puramente «smanettone»: «non so cosa ho fatto, ma funziona»
- **NO**: essere fuori dal contesto del corso: seguire il percorso proposto con le proprie tempistiche

- **SÌ**: mettere in pratica le nozioni apprese
- **SÌ**: cimentarsi con un problema senza cercare di adattare soluzioni altrui («scopiazzare» in Rete)
- **SÌ**: applicare una metodologia di programmazione
- **SÌ**: frequentare i laboratori e svolgere gli esercizi
- **SI'**: attenzione, concentrazione, precisione (i dettagli contano!)
- **SI'**: atteggiamento attivo/responsabile (il programma fa tutto e solo quello che il programmatore scrive)
- **SI'**: uso intelligente degli strumenti disponibili (materiale, rete, compilatore, debugger, etc.)
- **SI'**: **tempo** dedicato: studio e pratica
- **SI'**: preparazione adeguata in vista dell'esame

Obiettivo del corso

Successo del corso misurato:

- non in base al numero di bocciati
- ma in base al numero di superi.

Obiettivo: portare il maggior numero di voi a superare la soglia di conoscenze/competenze ritenuta necessaria.

Modalità di erogazione e di interazione

Lezioni ed esercitazioni (tutte in presenza) (4.5 h/sett):

- non è previsto lo streaming in Virtual Classroom
- sul Portale sono disponibili le videoregistrazioni delle lezioni dell'a.a. 2021/22 fruibili in asincrono

Laboratori (tutti in presenza) (1.5 h/sett):

- non è previsto lo streaming in Virtual Classroom, né la disponibilità di videoregistrazioni

Laboratorio

- 10 laboratori raggruppati in 3 gruppi
- testo pubblicato sul Portale la settimana prima
  - esercizi propedeutici/opzionali
  - esercizi valutati
- da svolgere a casa e nelle ore in laboratorio individualmente, non in gruppo
- per ognuno dei 3 gruppi di laboratori:
  - caricamento sul Portale entro le date indicate di tutti gli esercizi valutati (se ne tiene conto in valutazione):
  - obbligatorio per chi vuole usufruire del bonus (solo chi ha ASD in carico come nuovo frequentante sin dall'inizio del semestre)
- bonus di max 2/30 incluso nel voto complessivo a valle dell'orale. Bonus valido sino al 30 settembre 2024

Assistenza in Laboratorio

I corso:

- Andrea Protopapa

Riccardo Zaccone

II corso

- Leonardo Iurada

Luca Robbiano

III corso

- Paolo Rabino

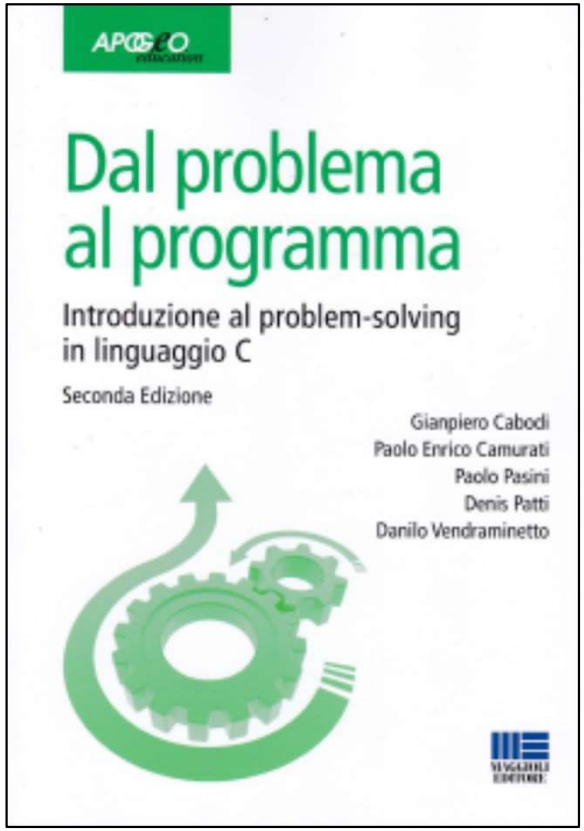
Simone Peirone

Testi fortemente consigliati

Problem-solving elementare in C:

G. Cabodi, P. Camurati, P. Pasini, D. Patti, D. Vendraminetto:
Dal problema a programma: introduzione al problem-solving in linguaggio C
II ed., Apogeo, 2016

Già consigliato per Tecniche di Programmazione



Ricorsione e problem-solving

G. Cabodi, P. Camurati, P. Pasini, D. Patti, D. Vendraminetto: *Ricorsione e problem-solving: strategie algoritmiche in linguaggio C,* Apogeo, 2015

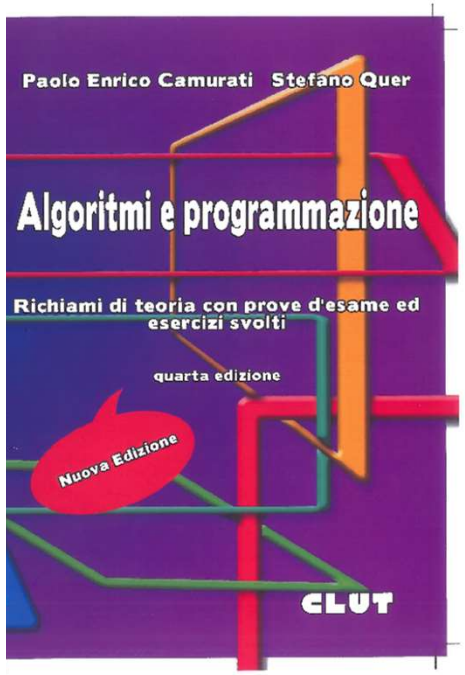
Puntatori, strutture dinamiche, modularità

G. Cabodi, P. Camurati, P. Pasini, D. Patti, D. Vendraminetto: *Puntatori e strutture dati dinamiche: allocazione della memoria e modularità in linguaggio C* Apogeo, 2016

Esercizi di teoria con richiami

P. Camurati, S. Quer:
Algoritmi e Programmazione:
richiami di teoria con esercizi svolti
IV edizione , CLUT, 2017

Comprende anche argomenti svolti a Tecniche di Programmazione

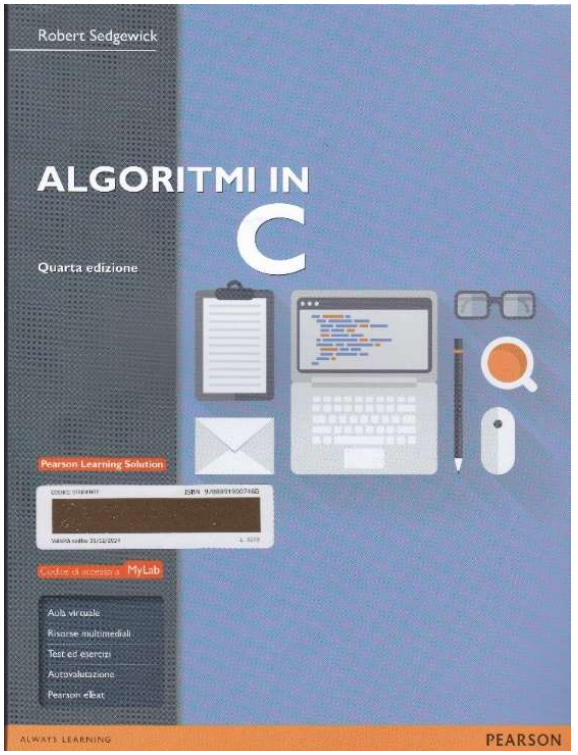


Esercizi svolti e commentati

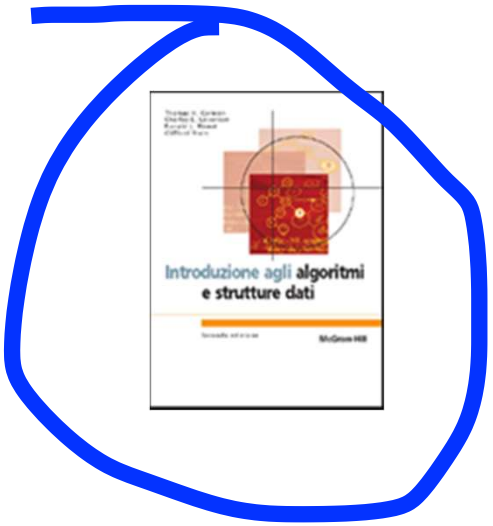
G. Cabodi, P. Camurati, P. Pasini, D. Patti, D. Vendraminetto: *Algoritmi e programmazione in pratica: da specifiche a codice C* Apogeo, 2018

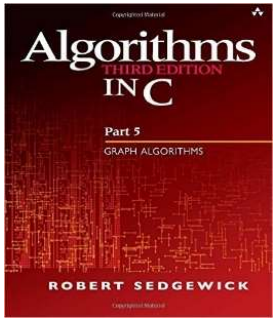
Altri testi

- R. Sedgewick, *Algoritmi in C (con MyLab – extext),* IV edizione, Pearson, 2015
- P.J. Deitel, H.M. Deitel *Il linguaggio C. Fondamenti e tecniche di Programmazione 8/Ed.* Pearson

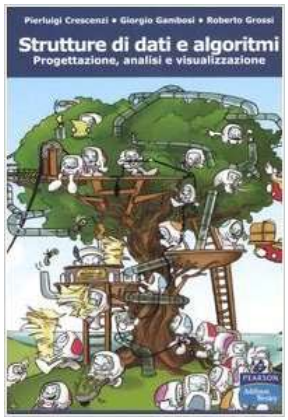


- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduzione agli algoritmi e strutture dati,* seconda edizione, McGraw Hill 2005
- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms, fourth edition*, The MIT Press, 2022
- R. Sedgewick, *Algorithms in C,* 3rd edition, Part 5: Graph Algorithms, Addison-Wesley, 2002





- P. Crescenzi, G. Gambosi, R. Grossi, *Strutture di dati e algoritmi*, Pearson Addison-Wesley 2006
- A. Bertossi, A. Montresor, *Algoritmi e strutture di dati*, III ediz., Città Studi edizioni, 2014
- R. Sedgewick, K. Wayne, *Algorithms Part I & II,* www.coursera.org



Materiale

Attraverso il portale della didattica verrà reso disponibile il seguente materiale relativo al corso:

- lezioni videoregistrate dell'a.a. 2021/22
- materiale usato in lezioni, esercitazioni e laboratori
- regole d'esame
- programma del corso

Sul portale compariranno anche orari, temi d'esame, risultati, avvisi.

Per ragioni di prestazioni, è possibile che parte del materiale, in paricolare le videolezioni, venga reso disponbile anche su Dropbox.

Esame

all'esame molto probabilmente non ci sarà tempo di finire e debuggare il programma, perchè è tanto difficile.

L'esame si compone di:

- una prova scritta:
  - durata massima 2h40
  - parte di teoria: durata massima 60 min, esercizi e risposte su carta a domande teoriche, punteggio massimo 12 punti
  - parte di programmazione C: durata massima 100 min, esercizi in 2 modalità
- un esame orale

Esame scritto ed esame orale sono obbligatori.

Lo scritto si svolgerà in aula su PC con lockdown browser. In caso di problemi tecnici è sempre garantita la modalità su carta.

La parte di programmazione C è disponibile in 2 modalità:

1. orientata al progetto: progettazione e realizzazione di un programma in grado di risolvere un problema. Punteggio: massimo 18 punti
2. orientata alla padronanza del C avanzato (puntatori, allocazione dinamica, ricorsione), delle strutture dati e degli algoritmi fondamentali. Punteggio: massimo 12 punti.

sommato ai 12 della parte di teoria fa 24…

NB: la preparazione richiesta è identica.

Materiale consultabile: **NON** è possibile consultare testi, appunti, dispense, etc., **NON** è possibile utilizzare supporti di tipo elettronico (cellulari, palmari, etc.) per comunicare.

Correzione dello scritto

Scritto di programmazione:

- si riceverà in automatico una copia del proprio elaborato
- si dovrà:
  - verificare la correttezza e la funzionalità del programma
  - caricare sul Portale il seguente materiale (entro tre giorni dalla data dello scritto):
    - relazione (max 3 pagine) sulla soluzione adottata (strutture dati, algoritmo, etc.)
    - copia del programma corretto, con evidenziate le modifiche rispetto al programma consegnato.

Qualora lo/la studente non carichi il materiale indicato entro la data prevista, la prova scritta (teoria e programmazione) non viene corretta.

Ammissione all'orale

votoTeoria≥soglia1 && votoC ≥soglia2 && votoTeoria+votoC ≥15/30

soglia1 e soglia2 definite di volta in volta in funzione della difficoltà
tra 5 e 7

Esame orale

- Gli orali si svolgono indicativamente 7-10 giorni dopo lo scritto
- Teoria: mira ad accertare le conoscenze teoriche acquisite in termini di comprensione dei concetti e di loro esposizione

  NB: non basta imparare meccanicamente come si risolvono gli esercizi della parte di Teoria
- Programmazione C:
  - realizzazione e manipolazione di strutture dati mediante funzionalità avanzate del linguaggio
  - implementazione di varianti di algoritmi visti a Teoria

  NB: bisogna saper scrivere del codice in tempo reale senza tentennamenti.
- Laboratorio: domande sugli esercizi consegnati

no essercizi completi, serve a vedere se conferma le abilita dello scritto

possono chiedere algoritmi presentati in classe, che non chiedono allo scritto

In conclusione:

non basta preparare l'orale in pochi giorni dopo la pubblicazione dei voti dello scritto appiccicando qualche nozione o memorizzando qualche funzione C.

L'orale è la sintesi della preparazione svolta durante il corso.