Principali informazioni sull'insegnamento		
Denominazione dell'insegnamento	**Programmazione**	
Corso di studio	Informatica	
Anno Accademico	2025/26	
Crediti formativi universitari (CFU) / European Credit Transfer and Accumulation System (ECTS)	12 CFU	
Settore Scientifico Disciplinare	INF/01	
Lingua di erogazione	Italiano	
Anno di corso	Primo	
Periodo di erogazione	1^ semestre, le date esatte sono riportate nel manifesto/regolamento	
Obbligo di frequenza	La frequenza è fortemente raccomandata	
Sito web del corso di studio	https://www.uniba.it/it/corsi/informatica/corso-di-laurea-in-informatica/	
Docente/i		
Nome e cognome	Pasquale Ardimento	
Indirizzo mail	pasquale.ardimento@uniba.it	
Telefono		
Sede	Dipartimento di Informatica, Via Orabona 4, 70125, Bari. Stanza n.569, V piano.	
Sede virtuale	Piattaforma ADA - https://elearning.di.uniba.it/	
Sito web del docente	https://www.uniba.it/it/docenti/ardimento-pasquale/	
Ricevimento	Martedì 14:30 – 15:30, o in altro giorno previo appuntamento per e-mail	
Syllabus		
Obiettivi formativi	Il corso si propone di introdurre la metodologia del problem solving e gli elementi base della programmazione imperativa strutturata per formulare soluzioni algoritmiche a problemi di varia complessità. In particolare, lo studente acquisirà la capacità di applicare le tecniche del problem solving, di individuare strategie di soluzione di natura algoritmica e di usare il linguaggio di programmazione C per implementarle.	
Prerequisiti	Non è richiesto alcun prerequisito in quanto trattasi di un insegnamento del primo semestre del primo anno.	
Contenuti di insegnamento (Programma)		

Mod	**Argomenti** (l'indicazione oraria si intende stimata)	**Ore**
1	Presentazione del corso, contenuti, modalità d'esame, frequenza alle lezioni, orari, modalità di esercitazione in aula, ecc. Introduzione alla Programmazione: dal problema al programma	2
2	L'algoritmo come astrazione. Le proprietà dell'algoritmo	2
3	Il Problem Solving e la fase di analisi L'analisi del problema. La formulazione del problema e le sue specifiche. I dati del problema. I risultati attesi. Gli errori e i casi limite. Ambiguità, limiti dei valori e controlli dei dati. Esempi ed esercizi.	5

		4	Il Problem Solving e la fase di progettazione La scomposizione del problema in sottoproblemi. Le tecniche: topdown, bottom-up, ibrida. La costruzione del metodo solutivo. La rappresentazione dell'algoritmo mediante linguaggi di rappresentazione: i diagrammi di flusso strutturati, il linguaggio lineare, l'albero di decomposizione, i	10

			diagrammi di Nassi-Schneidermann. Il teorema di Bohm-Jacopini: enunciato. Esempi ed esercizi.	
		5	Problemi semplici e problemi complessi. Scomposizione sequenziale, selettiva, iterativa e ricorsiva. Esempi ed esercizi.	4
		6	Il programma. Le variabili. Tipi di istruzione. Istruzioni dichiarative. Istruzioni di ingresso-uscita. Istruzione di assegnazione. Espressioni aritmetiche. Espressioni logiche. Costanti. Strutture di controllo. Esempi ed esercizi.	4
		7	Algoritmi elementari: conteggio, sommatoria e media di un insieme di numeri, fattoriale, conversione da caratteri a numeri in base 10, numero primo, massimo comun divisore, serie di Fibonacci, scambio.	4
		8	Tipi di dati. La dichiarazione di tipo. Tassonomia dei tipi di dato. Tipi standard. Tipi semplici definiti dall'utente: enumerativo, subrange.	4
		9	Tipi di dati. Tipi strutturati. Dato strutturato e tipo Array. Struttura di dati e tipo Record. Rappresentazione interna degli array e dei record. Gli indici degli array. Esempi ed esercizi.	6
		10	Esercitazione e preparazione alla prova di autovalutazione: analisi di problemi, scomposizione in sottoproblemi, progettazione algoritmica della soluzione. Attività partecipativa cooperativa svolta con l'assistenza del docente	2
		11	Esercitazione per autovalutazione: risoluzione di un problema secondo le fasi del problem solving escluso la codifica.	3
		12	Auto correzione prova di autovalutazione assistita dal docente	1
		13	La programmazione modulare. I sottoprogrammi. Utilità dei sottoprogrammi. Sottoprogramma come astrazione funzionale. Struttura risultante di un programma. La chiamata dei sottoprogrammi. Nidificazione. Esempi.	6
		14	La comunicazione dei sottoprogrammi con l'ambiente esterno e l'ambiente chiamante. La vista di un sottoprogramma. L'effetto shadowing. Variabili globali, locali e non locali. Le regole di visibilità. L'ambito e la durata delle variabili. Esempi ed esercizi.	2
		15	I sottoprogrammi. Allocazione statica e allocazione dinamica. Il side effect. Il contour model. Esempi.	2
		16	Sottoprogrammi. I parametri effettivi e i parametri formali. Modalità di passaggio dei parametri: per valore, per referenza e per nome. Valutazione e confronto tra le modalità di passaggio. Esempi ed esercizi.	3
		17	Sottoprogrammi. Le procedure. Intestazione e chiamata delle procedure. Esempi ed esercizi.	2
		18	Sottoprogrammi. Le funzioni. Intestazione e chiamata delle funzioni. Procedure vs funzioni. Esempi ed esercizi.	3

	Lezioni frontali, esercitazioni ed attività autonome e di gruppo in aula e a casa (come dettagliato nel programma). Gli studenti non frequentanti possono lavorare singolarmente prendendo accordi con il docente.			
		19	Sottoprogrammi. Sottoprogrammi come parametri. Attivazione dei sottoprogrammi. I record di attivazione e lo stack. La concatenazione statica e la concatenazione dinamica. La gestione dell'esecuzione. Esempi ed esercizi.	3
		20	La ricorsione. Funzioni ricorsive. Gestione dell'esecuzione con le funzioni ricorsive. Esempi ed esercizi.	4
		21	Algoritmi fondamentali: Algoritmi su array: stampa istogrammi mediante array, elimina duplicati su array ordinati. Algoritmi su matrici. Ricerca lineare. Ricerca binaria. Fusione di array. Esempi ed esercizi	3
		22	Algoritmi di ordinamento: sort per selezione, per scambio, per inserzione. Algoritmi ricorsivi: Fattoriale.	8
		23	Struttura dei programmi C. Tipi di dati: semplici predefiniti –int, float, double e char-, enumerazione esplicita dei valori. Definizione di tipo.	3
		24	Tipi strutturati in C: il costruttore array, il costruttore struct ed il costruttore puntatore. Compatibilità dei tipi.	4
		25	Strutture di controllo in C. Istruzioni di selezione: If, Switch; Istruzioni iterative: while, do-while, for.	4
		26	Funzioni e procedure in C: definizione, chiamata, prototipo, passaggio dei parametri.	4
		27	Ambito di visibilità delle variabili in C. Array come parametri. Strutture come parametri. Esempi ed esercizi.	4
		28	Effetti collaterali. Procedure e funzioni predefinite. Standard library. Puntatori. File.	5
		29	Strutture dati: pila, coda, lista. Esempi ed esercizi.	12
		30	Installazione, configurazione e uso dell'ambiente di sviluppo.	4
Testi di riferimento	Testi da cui studiare: C How to Program, Ninth Edition, P. Deitel e H. Deitel Gli studenti che lo desiderano possono ottenere il testo in prestito dalla Biblioteca. Può convenire verificarne la disponibilità mediante il Sistema Bibliotecario di Ateneo https://opac.uniba.it/easyweb/w8018/index.php? e contattare la biblioteca per concordare il prestito.			
Note ai testi di riferimento	Nel corso delle lezioni la docente utilizzerà delle slide che sono disponibili sulla piattaforma ADA del dipartimento (v. sopra 'sede virtuale'). Sulla piattaforma sono anche disponibili alcune prove scritte di esami svolte.			

Organizzazione della didattica

Ore

Totali	Didattica frontale	Laboratorio ed esercitazioni	Studio individuale
300 ore	72 ore	45 ore	183 ore

CFU/ETCS

12 CFU	9 CFU	3 CFU	

Metodi didattici

Risultati di apprendimento previsti	
Conoscenza e capacità di comprensione	• Acquisire la conoscenza degli aspetti teorici e pratici relativi alla progettazione delle soluzioni di problemi (problem solving) mediante l'uso del computer e la conoscenza degli strumenti che si utilizzano per la programmazione. • Acquisire conoscenze che consentano allo studente di comprendere come si può indicare ad un elaboratore elettronico (macchina automatica di impiego universale, hardware) la soluzione di un problema o di una classe di problemi, che l'elaboratore può risolvere, con un metodo ed un linguaggio appropriato, creando un apposito programma (software) eseguibile dall'elaboratore. • Acquisire la capacità di ragionare ed individuare una soluzione ad un problema (algoritmo) secondo il paradigma della programmazione imperativa strutturata.
Conoscenza e capacità di comprensione applicate	• Comprendere l'uso di un linguaggio di progettazione non convenzionale (es. • pseudocodice) e l'uso di una rappresentazione grafica (es. flow chart) per descrivere con un formalismo semplice un algoritmo; • Comprendere il lessico, la sintassi e la semantica del linguaggio di programmazione C; • Acquisire la capacità di scrivere un programma strutturato in linguaggio C; • Acquisire la capacità di individuare casi di test per il dominio cui fa riferimento il programma creato; • Acquisire la capacità di utilizzare un ambiente di sviluppo per trasformare il programma sorgente (in C) in programma eseguibile ed eseguirlo.
Competenze trasversali	**Autonomia di giudizio** • Acquisire la capacità di verificare che l'algoritmo individuato risponda alle specifiche di un problema; • Saper valutare e interpretare in maniera autonoma diverse strategie risolutive analizzando gli algoritmi proposti e fornendo soluzioni alternative. • Acquisire la capacità di verificare che i risultati ottenuti dopo l'esecuzione del programma siano quelli attesi. **Abilità comunicative** • Imparare a commentare il codice prodotto al fine di renderlo comprensibile e agevolmente modificabile da altri professionisti, con l'obiettivo di sviluppare in team. **Capacità di apprendere in modo autonomo** • Capacità di approfondire concetti attraverso lo studio autonomo di materiale didattico bibliografico anche attraverso piattaforme di e-learning. • Capacità di completare autonomamente il percorso formativo previsto dal testo di riferimento, oltre i contenuti previsti dal programma dell'insegnamento. • Capacità di riutilizzare le conoscenze acquisite sia in situazioni problematiche nuove, sia in contesti nuovi di programmazione imperativa.
Valutazione	
Modalità di verifica dell'apprendimento	L'esame si articola in due prove, entrambe **obbligatorie** e da svolgere nello **stesso appello**: **Prova di laboratorio** • **Durata**: 60 minuti • **Attività**: realizzazione di un programma in linguaggio **C**, suddiviso in 4 punti. • **Valutazione**: ogni punto svolto correttamente (cioè con output conforme alle specifiche) assegna **3 punti**, in caso contrario **0**. • **Punteggio massimo**: 12 punti. **Prova scritta**

	• **Durata**: 60 minuti • **Attività**: risposta a **3 domande aperte**. • **Valutazione**: ciascuna domanda, se correttamente e completamente svolta, vale **6 punti**. • **Punteggio massimo**: 18 punti. **Regole generali** • Per il superamento dell'esame è necessario sostenere **entrambe le prove** e ottenere un punteggio complessivo pari ad almeno **18/30**. • In caso di punteggio insufficiente, sarà necessario ripetere **entrambe le prove**. • Il voto finale corrisponde alla **somma dei punteggi** delle due prove, fino a un massimo di **30/30**. **Valutazione.** Il voto finale è espresso in trentesimi (0–30), con possibilità di lode

Criteri di valutazione	• **Conoscenza e capacità di comprensione:** Lo studente: o dovrà essere in grado di analizzare problemi formulando anche ipotesi aggiuntive, individuando i dati necessari e sufficienti per la soluzione e fornendone la descrizione; o dovrà essere in grado di individuare una strategia di soluzione che prevede la scomposizione del problema in sottoproblemi e di saper rappresentare sia la scomposizione sia gli algoritmi con adeguati linguaggi di descrizione; o dovrà dimostrare di saper implementare la soluzione proposta utilizzando il linguaggio imperativo di riferimento e di saperla testare su campioni di dati; o dovrà essere in grado di generalizzare soluzioni per una classe di problemi con lo stile della programmazione strutturata. **Conoscenza e capacità di comprensione applicate:** Lo studente: o dovrà essere in grado di individuare una strategia di soluzione che prevede la scomposizione del problema in sottoproblemi e di saper rappresentare sia la scomposizione sia gli algoritmi con adeguati linguaggi di descrizione; o dovrà dimostrare di saper implementare la soluzione proposta utilizzando il linguaggio imperativo di riferimento e di saperla testare su campioni di dati. • **Autonomia di giudizio:** Lo studente: o dovrà essere in grado di correggere e validare il corretto funzionamento dei programmi sviluppati; o dovrà essere in grado di discutere le soluzioni proposte chiarendo le scelte progettuali e implementative. • **Abilità comunicative:** Lo studente: o dovrà essere in grado di rendere il codice scritto comprensibile ad altri, mediante la sua descrizione generale e commenti specifici alle istruzioni e alle strutture di controllo utilizzate; o dovrà dimostrare di aver acquisito piena conoscenza dei concetti presentati a lezione nonché degli algoritmi fondamentali. • **Capacità di apprendere:** Lo studente: o dovrà essere in grado di trasformare autonomamente algoritmi descritti con flowchart in programmi in linguaggio C; o dovrà essere in grado di utilizzare le soluzioni alternative descritte nel testo di riferimento, se non descritte nel corso delle lezioni, come ad esempio le diverse modalità di dichiarazione delle variabili.

<table>
<tr><td>Criteri di misurazione dell'apprendimento e di attribuzione del voto finale</td><td>
<table>
<tr><th>Voto</th><th>Descrittori</th></tr>
<tr><td>< 18
insufficiente</td><td>Conoscenze frammentarie e superficiali dei contenuti, errori nell’applicare i concetti, descrizione carente.</td></tr>
<tr><td>18 - 20</td><td>Conoscenze dei contenuti sufficienti ma generali, descrizione semplice, incertezze nell’applicazione di concetti teorici.</td></tr>
<tr><td>21 - 23</td><td>Conoscenze dei contenuti appropriate ma non approfondite, capacità di applicare i concetti teorici, capacità di presentare i contenuti in modo semplice.</td></tr>
<tr><td>24 - 25</td><td>Conoscenze dei contenuti appropriate ed ampie, discreta capacità di applicazione delle conoscenze, capacità di presentare i contenuti in modo articolato.</td></tr>
<tr><td>26 - 27</td><td>Conoscenze dei contenuti precise e complete, buona capacità di applicare le conoscenze, capacità di analisi, descrizione chiara e corretta.</td></tr>
<tr><td>28 - 29</td><td>Conoscenze dei contenuti ampie, complete ed approfondite, buona applicazione dei contenuti, buona capacità di analisi e di sintesi, descrizione sicura e corretta.</td></tr>
<tr><td>30
30 e lode</td><td>Conoscenze dei contenuti molto ampie, complete ed approfondite, capacità ben consolidata di applicare i contenuti, ottima capacità di analisi, di sintesi e di collegamenti interdisciplinari, padronanza di descrizione.</td></tr>
</table>
</td></tr>
<tr><td>Altro</td><td>Si suggerisce agli studenti di affidarsi esclusivamente alle informazioni/comunicazioni fornite sui siti ufficiali del Dipartimento di Informatica, ovvero sui gruppi social solo se costituiti e amministrati esclusivamente dai docenti dei relativi insegnamenti:

• https://www.uniba.it/it/ricerca/dipartimenti/informatica/didattica/corsi-dilaurea/corsi-di-laurea
• https://www.uniba.it/it/ricerca/dipartimenti/informatica
• https://elearning.di.uniba.it/

I programmi degli insegnamenti sono disponibili qui:
• https://programmi.di.uniba.it/

Le informazioni che tutti gli studenti dovrebbero conoscere sono scritte nei Regolamenti didattici e manifesti degli studi disponibili nel sito:
• https://www.uniba.it/it/ricerca/dipartimenti/informatica/didattica/corsi-dilaurea/corsi-di-laurea

Si suggerisce agli studenti di diffidare delle informazioni e dei materiali circolanti su siti o gruppi social non ufficiali, poiché spesso sono risultati non affidabili, non corretti o incompleti. Per ogni dubbio, chiedere un incontro al docente secondo le modalità previste per il ricevimento.

Link al corso sulla piattaforma e-learning del dipartimento ADA:
• https://elearning.di.uniba.it/</td></tr>
</table>

Main information on the course		
Course name	Programmazione	
Degree	Informatica	
Academic year	2025/26	
European Credit Transfer and Accumulation System (ECTS), in Italian Crediti Formativi Universitari (CFU)	12 CFU, 9 T1 + 3 T2 (each CFU corresponds to 25 hours (h) of student's time); CFU are of type T1, T2 or T3 T1 = 8 h lecture + 17 h individual study T2 = 15 h practice + 10 h individual study T3 = 25 h individual study	
Settore Scientifico Disciplinare		
Course language	Italian	
Course year	First	
Course period	First Semester - exact dates can be found in the didactic regulations	
Course attendance requirement	None, but it is highly recommended to attend classes	
Website of the Degree	https://www.uniba.it/it/corsi/informatica/corso-di-laurea-in-informatica/	

Teacher(s)	
Name and Surname	Pasquale Ardimento
email	pasquale.ardimento@uniba.it
phone	
office	Department of Informatics, Via Orabona 4, 70125, Bari. Room n.569, V floor.
e-learning platform	ADA platform - https://elearning.di.uniba.it/
Teacher's homepage	https://www.uniba.it/it/docenti/ardimento-pasquale
Office hours	Wednesday 2.30pm – 3.30pm, or on another day by appointment by email

Syllabus	
Course goals	The course aims to introduce the methodology of problem-solving and the basic elements of structured imperative programming to formulate algorithmic solutions to problems of varying complexity. In particular, the student will acquire the ability

	to apply problem-solving techniques, identify algorithmic solution strategies, and use the C programming language to implement them.
Prerequisites/requirements	No prerequisites are required, as this is a course taught in the first semester of the first year.
Course program	

Mod	Topics (estimated hours)	Hours
1	Course presentation, contents, exam modalities, attendance, schedule, in-class exercises, etc. Introduction to Programming: from problem to program	2
2	The algorithm as an abstraction. Properties of the algorithm	2
3	Problem Solving and the analysis phase Problem analysis. Problem formulation and its specifications. Problem data. Expected results. Errors and edge cases. Ambiguities, value limits and data validation. Examples and exercises.	5
4	Problem Solving and the design phase Breaking down the problem into subproblems. Techniques: top-down, bottom-up, hybrid. Building the solution method. Algorithm representation through representation languages: structured flowcharts, linear notation, decomposition tree, Nassi–Shneiderman diagrams. The Böhm–Jacopini theorem: statement. Examples and exercises.	10
5	Simple problems and complex problems. Sequential, selective, iterative, and recursive decomposition. Examples and exercises.	4
6	The program. Variables. Types of instructions. Declarative instructions. Input-output instructions. Assignment statement. Arithmetic expressions. Logical expressions. Constants. Control structures. Examples and exercises.	4
7	Elementary algorithms: counting, summation and average of a set of numbers, factorial, conversion from characters to numbers in base 10, prime number, greatest common divisor, Fibonacci series, swapping.	4
8	Data types. Type declaration. Taxonomy of data types. Standard types. Simple user-defined types: enumerated, subrange.	4
9	Data types. Structured types. Structured data and Array type. Data structure and Record type. Internal representation of arrays and records. Array indices. Examples and exercises.	6
10	Exercise and preparation for the self-assessment test: problem analysis, decomposition into subproblems, algorithmic design of the solution. Cooperative participatory activity with teacher assistance.	2

	11	Self-assessment exercise: solving a problem according to the problem-solving phases, excluding coding.	3
	12	Self-correction of the self-assessment test with teacher assistance.	1
	13	Modular programming. Subprograms. Usefulness of subprograms. Subprogram as functional abstraction. Resulting structure of a program. Subprogram calls. Nesting. Examples.	6
	14	Communication between subprograms and the external/calling environment. Subprogram scope. Shadowing effect. Global, local, and non-local variables. Visibility rules. Variable scope and lifetime. Examples and exercises.	2
	15	Subprograms. Static allocation and dynamic allocation. Side effects. The contour model. Examples.	2
	16	Subprograms. Actual and formal parameters. Parameter passing methods: by value, by reference, and by name. Evaluation and comparison of passing methods. Examples and exercises.	3
	17	Subprograms. Procedures. Procedure headers and calls. Examples and exercises.	2
	18	Subprograms. Functions. Function headers and calls. Procedures vs functions. Examples and exercises.	3
	19	Subprograms. Subprograms as parameters. Subprogram activation. Activation records and the stack. Static and dynamic chaining. Execution management. Examples and exercises.	3
	20	Recursion. Recursive functions. Execution management with recursive functions. Examples and exercises.	4
	21	Fundamental algorithms: Algorithms on arrays: histogram printing with arrays, removing duplicates from sorted arrays. Algorithms on matrices. Linear search. Binary search. Array merging. Examples and exercises.	3
	22	Sorting algorithms: selection sort, exchange sort, insertion sort. Recursive algorithms: Factorial.	8
	23	Structure of C programs. Data types: predefined simple types – int, float, double, and char –, explicit enumeration of values. Type definition.	3

	24	Structured types in C: array constructor, struct constructor, pointer constructor. Type compatibility.	4
	25	Control structures in C. Selection statements: If, Switch; Iterative statements: while, do-while, for.	4
	26	Functions and procedures in C: definition, call, prototype, parameter passing.	4
	27	Variable scope in C. Arrays as parameters. Structures as parameters. Examples and exercises.	4
	28	Side effects. Predefined procedures and functions. Standard library. Pointers. Files.	5
	29	Data structures: stack, queue, list. Examples and exercises.	12
	30	Installation, configuration, and use of the development environment.	4

Books of reference	Books for studying: C How to Program, Ninth Edition, P. Deitel e H. Deitel Students who wish can borrow the books from the Library. It may be useful to check availability through the University Library System at https://opac.uniba.it/easyweb/w8018/index.php and contact the library to arrange the loan.
Notes to the books	During the lessons the teacher will use slides that retrace the contents of the book, therefore, they will not be provided. The reference text contains all the topics of the course, therefore, it is advisable to study from the text and to carry out independently and constantly all the exercises included at the end of each chapter covered in class. On the department's e-learning platform (see above 'virtual office') are available: • supporting video material used in class; • some traces of written tests of exams, with examples of traces carried out;

Organization of the didactic activities

Hours

Total	Lectures	Practice sessions	Project work	Individual study
300 hours	72 hours	45 hours	00 hours	183 hours

CFU/ETCS

12 CFU	09 CFU	03 CFU	00 CFU	

Teaching methods

	Lectures, exercises and autonomous and group activities in the classroom and at home (as detailed in the program). Non-attending students can work individually by making arrangements with the teacher.

Expected learning outcomes	
Knowledge and understanding	• Acquire knowledge of the theoretical and practical aspects related to designing problem-solving solutions using a computer, as well as an understanding of the tools used for programming. • Gain an understanding of how to instruct a computer (a universal automatic machine, hardware) to solve a problem or a class of problems that it can handle, using an appropriate method and language, by creating a specific program (software) executable by the computer. • Develop the ability to reason and identify a solution to a problem (algorithm) according to the structured imperative programming paradigm. • Develop the ability to reason and identify a solution to a problem (algorithm) according to the Object-Oriented programming paradigm
Applying knowledge and understanding	• Understand the use of a non-conventional design language (e.g., pseudocode) and the use of a graphical representation (e.g., flowchart) to describe an algorithm with simple formalism. • Understand the lexicon, syntax, and semantics of the C programming language. • Develop the ability to write a structured program in C. • Acquire the ability to identify test cases for the domain to which the created program applies. • Gain the ability to use a development environment to transform the source program (in C) into an executable program and run it.
Other skills	*Making judgements* • Develop the ability to verify that the identified algorithm meets the problem specifications. • Be able to evaluate and interpret different solution strategies independently by analyzing proposed algorithms and providing alternative solutions. • Acquire the ability to check that the results obtained after executing the program are as expected. *Communication* • Learn to comment on the produced code to make it understandable and easily modifiable by other professionals, with the aim of developing in a team. *Learning skills* • Ability to deepen concepts through independent study of bibliographic educational material, including e-learning platforms. • Ability to independently complete the educational path outlined by the reference text, beyond the content covered by the course syllabus. • Ability to reuse acquired knowledge in both new problem situations and new contexts of imperative programming.

Assessment	
Assessment methods	The exam consists of two tests, both mandatory and to be taken in the same session: **Laboratory Test** • **Duration:** 60 minutes • **Activity:** development of a C program, divided into 4 tasks. • **Evaluation:** each correctly completed task (i.e., producing output consistent with the specifications) is worth 3 points; otherwise, 0 points. • **Maximum score:** 12 points. **Written Test** • **Duration:** 60 minutes • **Activity:** answer 3 open-ended questions. • **Evaluation:** each question, if correctly and fully answered, is worth 6 points. • **Maximum score:** 18 points. **General Rules** • To pass the exam, both tests must be taken and a total score of at least 18/30 must be achieved. • In case of an insufficient score, both tests must be retaken. • The final grade corresponds to the sum of the scores of the two tests, up to a maximum of 30/30. **Grading:** The final grade is expressed on a 30-point scale (0–30), with the possibility of *cum laude*.
Evaluation criteria	• Knowledge and understanding: - The student must be able to analyze problems, even formulating additional hypotheses, identifying the necessary and sufficient data for the solution, and providing a description. - They must be able to identify a solution strategy that involves breaking down the problem into sub-problems and representing both the breakdown and the algorithms with appropriate descriptive languages. - They must demonstrate the ability to implement the proposed solution using the reference imperative language and test it on data samples. - They must be able to generalize solutions for a class of problems using structured programming. • Applied knowledge and understanding: - They must be able to identify a solution strategy that involves breaking down the problem into sub-problems and representing both the breakdown and the algorithms with appropriate descriptive languages. - They must demonstrate the ability to implement the proposed solution using the reference imperative language and test it on data samples. • Autonomy of judgment:

	- The student must be able to correct and validate the correct functioning of the developed programs. - They must be able to discuss the proposed solutions, explaining the design and implementation choices. • Communication skills: - The student must be able to make the written code understandable to others by providing a general description and specific comments on the instructions and control structures used. - They must demonstrate full knowledge of the concepts presented in the lessons as well as the fundamental algorithms. • Learning skills: - The student must be able to independently transform algorithms described with flowcharts into C programs. - The student must be able to use alternative solutions described in the reference text, even if not covered during lessons, such as the different methods of declaring variables.
Measurements and final grade	• < 18 ○ Insufficient: Fragmented and superficial knowledge of the content, errors in applying concepts, inadequate description. • 18 - 20 ○ Sufficient: General knowledge of the content, simple description, uncertainty in applying theoretical concepts. • 21 - 23 ○ Adequate: Appropriate but not in-depth knowledge of the content, ability to apply theoretical concepts, ability to present content in a simple manner. • 24 - 25 ○ Good: Appropriate and broad knowledge of the content, reasonable ability to apply knowledge, ability to present content in a structured manner. • 26 - 27 ○ Very Good: Precise and complete knowledge of the content, good ability to apply knowledge, analytical skills, clear and correct description. • 28 – 29 ○ Excellent: Broad, complete, and in-depth knowledge of the content, good application of concepts, strong analytical and synthesis skills, confident and correct description. • 30 - 30 with honors ○ Outstanding: Very broad, complete, and in-depth knowledge of the content, well-established ability to apply concepts, excellent analytical, synthesis, and interdisciplinary connection skills, mastery in description.
Further information	It is recommended that students rely exclusively on the information and communications provided on the official websites of the Department of Computer Science or on social groups only if they are created and managed solely by the instructors of the respective courses: • https://www.uniba.it/it/ricerca/dipartimenti/informatica/didattica/corsi-dilaurea/corsi-di-laurea • https://www.uniba.it/it/ricerca/dipartimenti/informatica • https://elearning.di.uniba.it/ Course syllabi are available here: • https://programmi.di.uniba.it/

	Information that all students should be aware of is written in the Academic Regulations and study guides available on the website: • https://www.uniba.it/it/ricerca/dipartimenti/informatica/didattica/corsi-dilaurea/corsi-di-laurea Students are advised to be cautious of information and materials circulating on unofficial websites or social groups, as they are often unreliable, incorrect, or incomplete. For any doubts, please request a meeting with the instructor according to the office hours procedures. Link to the course on the department's ADA e-learning platform: • https://elearning.di.uniba.it/