LABORATORIO DI PROGRAMMAZIONE
SICUREZZA DEI SISTEMI E DELLE RETI INFORMATICHE
UNIVERSITÀ DEGLI STUDI DI MILANO

2025–2026

1.12.2025

Indice

Avvertenze. L'esame avrà una durata di 3.5h ed è diviso in tre parti, che peseranno quasi equamente sulla valutazione finale.

la prima parte, in linguaggio `C`, servirà a valutare le vostre competenze di base di programmazione. La seconda parte, in linguaggio `C`, e la terza, in linguaggio `Java`, serviranno a valutare le competenze specifiche sui due linguaggi.

Questa simulazione include due sotto-esami, ciascuno comprendente parte 1 e parte 2.

Si raccomanda di leggere interamente il tema d'esame con attenzione prima di cominciare a scrivere le soluzioni dei singoli esercizi, in modo da avere chiaro l'obiettivo finale.

Ultima revisione: 28 novembre 2025.

Sotto-Esame 1

Parte 1 - [Linguaggio C].

Esercizio 1 - Cifrario di Cesare - Cifratura.

Il cifrario di Cesare è uno dei più antichi algoritmi crittografici di cui si abbia traccia storica. È un cifrario a sostituzione monoalfabetica, in cui ogni lettera del testo in chiaro è sostituita, nel testo cifrato, dalla lettera che si trova k posizioni dopo nell'alfabeto. Questi tipi di cifrari sono detti anche cifrari a sostituzione o cifrari a scorrimento a causa del loro modo di operare: la sostituzione avviene lettera per lettera, scorrendo il testo dall'inizio alla fine. Per decifrare, si effettua l'operazione inversa, ovvero si sostituisce ogni lettera con quella a posizione $-k$.
Un esempio di cifratura, con $k = 3$, è visibile nella tabella qui sotto.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Implementate una funzione `cifra` che, ricevuta una stringa (e altri parametri necessari), ne restituisca la versione cifrata utilizzando una chiave k (intera).
Implementate poi la funzione duale `decifra`.
Testate il corretto funzionamento della funzione con dei casi di test appositi.

Parte 2 - [Linguaggio C].

Esercizio 2 - Cifrario di Cesare - Struttura Dati.

Partendo dall'esercizio definito nella Parte 1, riguardante il Cifrario di Cesare, definire una struttura dati `riga`, che contenga una stringa di al più n caratteri ed eventuali parametri necessari. La lunghezza massima della stringa N non è nota a priori.
Scrivere la funzione `leggiRiga` che legga da file una riga di un testo da cifrare e che salvi tale riga nella apposita struttura dati così definita.
La funzione `leggiRiga` dovrà salvare la stringa letta da file in una stringa dimensionata esattamente come il testo letto dalla riga n-esima del file (più il carattere terminatore). Ad esempio, se la dimensione massima della stringa è $n = 10$, e la stringa letta alla i-esima riga è `"CIAO"`, la dimensione della stringa salvata sarà di 5.

Esercizio 3 - Cifrario di Cesare - Main.

Scrivere la funzione `main` del programma per permettere all'utente di cifrare e decifrare un testo utilizzando il Cifrario di Cesare.

Le funzionalità che deve avere il `main`, e che devono essere svolte in maniera sequenziale, sono le seguenti:

- Scegliere la funzionalità da svolgere (cifrare o decifrare un testo);
- Inserire il nome del file da leggere;
- Inserire la chiave di cifratura;

- Leggere il file contenente il testo da cifrare/decifrare, utilizzando la struttura dati definita in Esercizio 2;
- Stampare a schermo il file decifrato/cifrato;
- Chiedere all'utente se vuole salvare il file decifrato/cifrato. In caso di risposta positiva, chiedere all'utente un nome di file su cui salvare il risultato;
- Salvare il testo cifrato/decifrato su file.

Al termine dell'operazione di cifratura/decifratura e salvataggio file, il programma termina. Gestire eventuali errori.
Il file da leggere conterrà, come primo elemento, un numero n che indica la lunghezza massima di ogni riga. Successivamente, conterrà il testo di riferimento, composto da un numero non noto a priori di righe, ognuna di lunghezza al più n. Assumete che il file sia strutturato in maniera corretta.
Il file da scrivere dovrà contenere solo il testo, cifrato o decifrato.
Testare il programma con un file creato da voi.

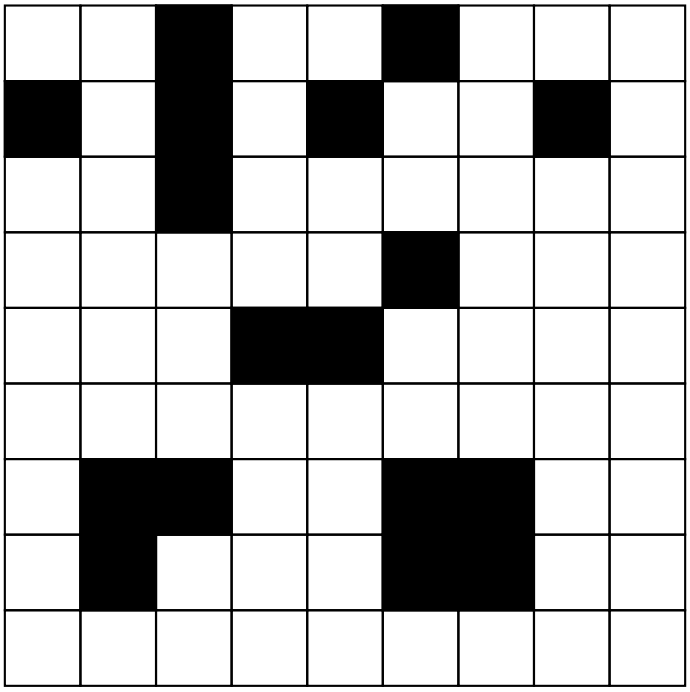


FIGURA 1. Un esempio del gioco della vita con una griglia 9×9. Una cella nera è una cella viva, una cella bianca è una cella morta.

SOTTO-ESAME 2

Parte 1 - [Linguaggio C].

Esercizio 4 - Il gioco della vita.

Il ***gioco della vita*** è un celebre automa cellulare che ha come scopo mostrare come comportamenti simili alla vita possano emergere da regole semplici e da interazioni di molti corpi. Tale gioco, sviluppato dal matematico britannico John Conway tra gli anni 60 e 70, è stato ampiamente studiato nell'ambito dell'informatica, della matematica, e dell'intelligenza artificiale.

Il gioco della vita è un gioco senza giocatori: la sua evoluzione è determinata dal suo stato iniziale, senza necessità di alcun input.

Il mondo è rappresentato come una griglia di dimensioni $N \times N$, in cui ogni valore della cella può essere vivo (1) o morto (0). Lo stato della griglia evolve in intervalli di tempo discreti, cioè scanditi in maniera netta. Gli stati di tutte le celle in un dato istante sono usati per calcolare lo stato delle celle all'istante successivo.

Tutte le celle del mondo vengono quindi aggiornate simultaneamente nel passaggio da un istante a quello successivo: passa così una generazione.

Lo stato di una cella alla generazione successiva $(t + 1)$ dipende quindi dallo stato dei suoi vicini alla generazione corrente (t). Le regole sono semplici, e sono illustrate in Figura 2:

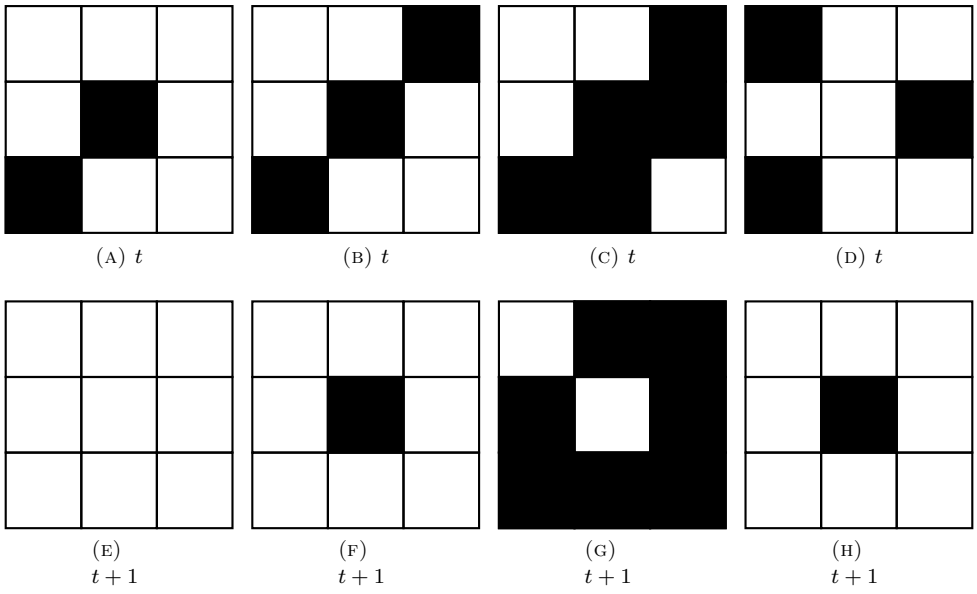
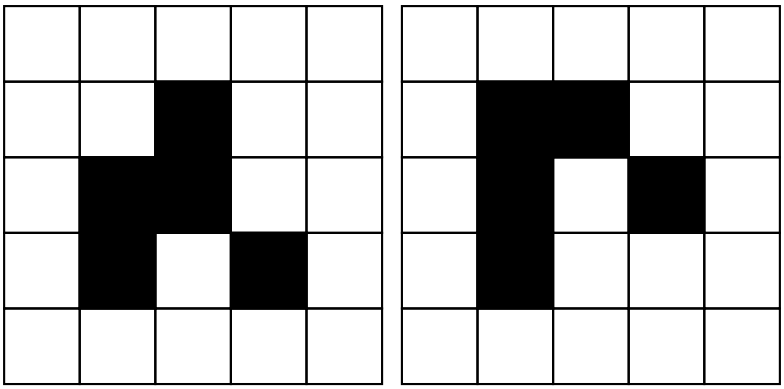


FIGURA 2. Lo stato della **cella centrale**, nella generazione successiva, dipende dallo stato dei suoi 8 vicini. Nel caso (A), la cella centrale morirà per isolamento, diventando (E). Nel caso (B) la cella centrale rimarrà viva (F). Nel caso (C) la cella centrale morirà per effetto di sovrappopolazione (G). Nel caso (D) la cella centrale, ora morta, sarà viva per effetoto di riproduzione (H).

- Una cella viva con meno di due celle vive vicine muore, per effetto dell'isolamento (Figura 2a);
- Una cella viva con due o tre celle vive vicine sopravvive alla generazione successiva (Figura 2b);
- Una cella viva con più di tre celle vive adiacenti muore, per effetto della sovrappopolazione (Figura 2c);
- Qualsiasi cella morta con esattamente tre celle vive adiacenti diventa una cella viva, per effetto di riproduzione (Figura 2d).

In Figura 3 trovate un esempio di una configurazione di una matrice 5×5 i due istanti successivi

Dovete implementare il gioco della vita in `C` e calcolare una serie di generazioni a partire da uno stato iniziale. Per il mondo considerate come riferimento una griglia di dimensione $N \times N$ con `N` definito a **compile time**, ad esempio $N = 20$. Considerate solo le celle all'interno della matrice. Le celle ai bordi avranno quindi meno vicini.



(A) t (B) $t+1$

FIGURA 3. Un esempio di due generazioni per una griglia 5×5.

Esercizio 4.1
Definite una funzione `stampa` che, ricevuto una matrice indicante uno stato del gioco della vita, ed eventuali parametri necessari, stampi a schermo in caratteri ASCII lo stato, indicando con una casella vuota (uno spazio) una cella morta (bianca), con un asterisco # una cella piena (nera) .

Ad esempio, lo stato indicato in Figura 1 sarà stampato nel seguente modo:

```
  #  #
# # #  #
  #
     #
   ##

 ##  ##
 #   ##
```

Esercizio 4.2
Definite una funzione `inizializza` che, ricevuto come input l'insieme delle celle, lo inizializzi in maniera casuale; per farlo, inizializzate ogni cella con un valore casuale, e convertitelo a vivo o morto. Utilizzate la funzione `rand()` definita in `stdlib.h`, che restituisce un intero casuale.

Esercizio 4.3
Definite una funzione `vicini` che, ricevuto come input lo stato corrente ed una cella, conti quanti sono i vicini della cella che sono vivi.

Esercizio 4.4
Definite una funzione `nuovaGenerazione` che, ricevuto come input lo stato corrente, generi la generazione successiva del gioco della vita.

Testate le quattro funzioni richieste con delle apposite funzioni di `test`; allegatele alla soluzione che consegnerete.

Il main che utilizza la struttura dati così definita è oggetto della Parte 2. Testate le singole funzioni, e considerate i possibili errori che potrebbero verificarsi durante

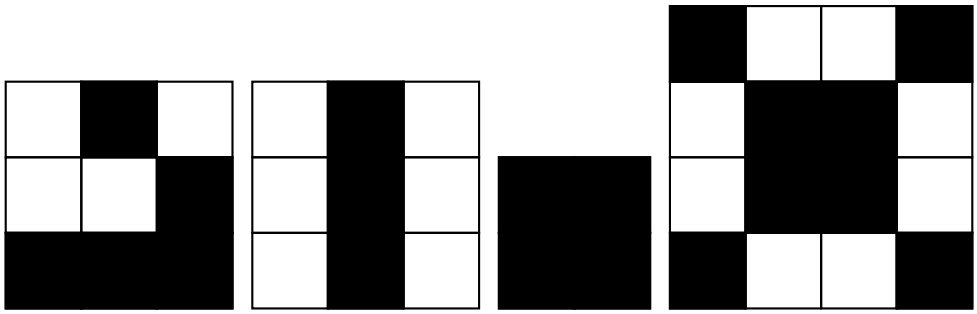


FIGURA 4. Quattro pattern di esempio. Il primo pattern (A), un *glider*, si muoverà con le generazioni. Il secondo pattern (B), un *oscillatore*, ha un comportamento transitorio. Il terzo ed il quarto pattern (C-D) producono, subito o in poche generazioni, delle configurazioni stabili.

l'esecuzione. In Figura 4 trovate alcuni esempi con delle configurazioni stabili da usare come casi di testing.

Parte 2 - [Linguaggio C].

Esercizio 5 - Il gioco della vita - Funzioni.

Esercizio 5.1
Definite una struttura dati `Pattern`, che rappresenta un pattern di dimensione $h \times w$, non nota a priori. La struttura dati deve contenere i valori di `h` e `w` della struttura dati, ed una matrice di dimensione concorde contenente il pattern.[1] Infine, gli ultimi due valori saranno le coordinate della cella nella griglia dove il pattern dovrà essere inserito. Esempi di pattern sono indicati in Figura 4.

Esercizio 5.2
Definite una funzione `inserisciPattern` che, ricevuto in ingresso lo stato corrente, un pattern, inserisca il pattern nella posizione data (sovrascrivendo eventuali dati già presenti).

Esercizio 5.3
Definite una funzione `caricaPatterns` che, ricevuto in ingresso il nome di un file e struttura dati adeguata, carichi dal file k patterns contenuti nel file, con k non noto a priori. La funzione restituisce il numero k di pattern letti. Il file sarà così strutturato: come primo elemento, il numero k di pattern al suo interno. Poi, per ognuno dei pattern, sono indicati il valore h e w della dimensione del pattern, seguito da $h \times w$ valori interi. Infine, viene indicata la cella di partenza dove il pattern deve essere inserito. Assumete che il file, se esiste, sia formattato correttamente. Ad esempio, il pattern relativo a Figura 4b sarà:

[1] Una matrice di dimensione $h \times k$ è rappresentabile da un vettore con $h \times k$ elementi. Scegliete la rappresentazione del dato più comoda per voi.

```
3 3
0 1 0 0 1 0 0 1 0
1 1
```

In questo caso il pattern viene inserito a partire da cella $(1-1)$.

mentre il pattern relativo a Figura 4c sarà:

```
2 2
1 1 1 1
3 4
```

In questo caso il pattern viene inserito a partire da cella $(3-4)$.
Attenzione: la funzione deve salvare tutti i pattern in una variabile adeguatamente strutturata dinamicamente, e passata come parametro alla funzione, non deve direttamente inserire i pattern nella matrice. Per questo dovrete usare la funzione `inserisciPattern` all'interno del `main`.

Esercizio 5.4
Scrivere un `main` che chieda all'utente il nome di un file ed il numero G di generazioni che vuole che siano generate. Il `main` deve leggere dal file indicato i pattern, definiti come all'esercizio 2.3, e deve successivamente utilizzarli per inizializzare lo status della matrice. Infine deve eseguire G generazioni del gioco della vita, stampandole a schermo in maniera sequenziale.

In caso di errori, il programma termina segnalandolo.

Esercizio 5.5 (Bonus)
Rendete ricorsiva la funzione `nuovaGenerazione`.

In caso di errori, il programma termina segnalandolo.

Istruzioni per la consegna

- Come consegnare:
  - Per gli esercizi in Linguaggio `C`, consegnate un unico file `.c` contenente il programma svolto per Parte 1 e Parte 2. Il file deve essere codice `C` valido (deve compilare).

  Commentare il codice per spiegare le scelte fatte è fortemente consigliato. Il codice **deve essere indentato e leggibile**.
- Sito per la consegna:

  `https://upload.di.unimi.it`
- Autenticarsi con le proprie credenziali di posta elettronica d'ateneo.
- Consegnare solo il codice sorgente (file `*.c`), NON l'eseguibile.
- Ogni file consegnato **non compilabile** non verrà corretto, anche se alcune sue parti sono giuste.