# APPELLO SCRITTO DI PROGRAMMAZIONE 1
## CORSO DI LAUREA IN MATEMATICA
## UNIVERSITÀ DEGLI STUDI DI MILANO
### 2017–2018
### 26.XI.2018

DIEGO VALOTA

INDICE



**Avvertenza.** Il tema d'esame richiede lo sviluppo di un singolo programma che implementi una versione semplificata di un automa cellulare ispirato al *Game of Life* del matematico britannico J.H. Conway. Le definizioni precise e i dettagli necessari sono dati nel testo. Svilupperete delle funzioni ausiliarie nel risolvere gli esercizi. Integrerete poi tutte le funzioni ausiliarie in un unico programma finale, che sarà costituito da un solo file sorgente. (Il tema d'esame richiede esplicitamente l'implementazione di alcune funzioni ausiliarie. È comunque sempre ammissibile aggiungere altre funzioni ausiliarie qualora lo riteniate opportuno per una migliore implementazione della vostra soluzione.) 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. I file di dati `dati1.txt` e `dati2.txt` allegati al testo d'esame vi serviranno per testare il vostro programma.

Ultima revisione: 26 novembre 2018.

## Premessa: descrizione di *Game of Life*.

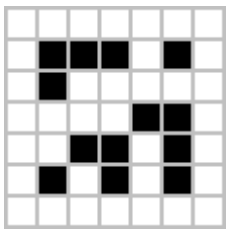


Figura 1. Una configurazione di un automa cellulare $7 \times 7$. Fonte: pagina Wikipedia del *Game of Life*.

La versione semplificata del *Game of Life* che implementerete, si gioca su una matrice di 12 righe per 12 colonne. La matrice rappresenta un'*automa cellulare*, dove ogni cella $(i, j)$ della matrice può essere *viva* o *morta*. Per ogni cella, le sue *vicine* sono le celle adiacenti in tutte le direzioni: orizzontale, verticale e diagonale. Ad esempio, per $0 < i, j < 11$, le 8 vicine della cella $(i, j)$ sono le celle $(i-1, j-1)$,$(i-1, j)$,$(i-1, j+1)$,$(i, j-1)$,$(i, j+1)$,$(i+1, j-1)$,$(i+1, j)$ e $(i+1, j+1)$.

Ogni cella interagisce con le sue vicine in accordo con le seguenti regole:

(1) se una cella viva ha meno di 2 vicine vive, allora muore (*scarsità*);
(2) se una cella viva ha 2 o 3 vicine vive, allora resta viva (*stabilità*);
(3) se una cella viva ha più di 3 vicine vive, allora muore (*sovrappopolazione*);
(4) se una cella morta ha esattamente 3 vicine vive, allora diventa viva (*riproduzione*).

**Attenzione:** quando una cella $(i, j)$ si trova lungo i *bordi* dell'automa, ovvero quando $i \in \{0, 11\}$ o $j \in \{0, 11\}$, in alcune direzioni non avrà delle vicine. In questi casi, per una corretta applicazione delle regole (1)–(4), assumiamo che le celle *esterne* all'automa siano morte. Ovvero che le vicine $(i', j')$ di $(i, j)$ tali che $i' < 0$ o $i' > 11$ o $j' < 0$ o $j' > 11$ siano morte.

Un automa passa da uno *stato* al successivo applicando le regole di cui sopra ad ogni cella **simultaneamente**. Il passaggio da uno stato al successivo è chiamato *transizione*. Si veda Fig. 2 per un esempio di transizione.

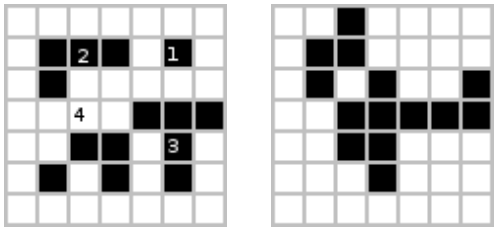



Figura 2. L'automa di Fig. 1 ed il suo stato successivo. I numeri che compaiono in alcune celle della prima figura, indicano la regola applicata per ottenere lo stato successivo della cella stessa. **Attenzione:** le regole sono state indicate solo per alcune celle dell'automa, per ottenere lo stato della figura a destra vanno applicate a tutte le celle dello stato a sinistra.

## Esercizio 1

**Griglia di gioco e visualizzazione.**

**Punti:** 7.

La griglia di gioco sarà rappresentata in memoria da una matrice `aut` di interi di dimensione $12 \times 12$. La matrice `aut` non può essere dichiarata globalmente.

Conveniamo che `aut[i][j]` varrà 0 quando la casella $(i, j)$ contiene una cella morta, varrà 1 quando la casella $(i, j)$ contiene una cella viva. Conveniamo che le celle della griglia `aut` siano rappresentate a video da un rettagnolo grigio chiaro quando morte e da un rettangolo grigio scuro quando vive.

Scrivete la funzione di visualizzazione `show()`, di prototipo opportuno, che visualizzi la griglia `aut`. Usate i codici *Unicode* `\u2591` e `\u2593` per stampare a schermo un rettangolo grigio chiaro ed uno grigio scuro rispettivamente (ad esempio: `printf("\u2591")` stamperà a schermo un rettangolo grigio chiaro). Il campo di gioco sarà quindi visualizzato sul terminale in modo simile a quanto mostrato in Fig. 3.

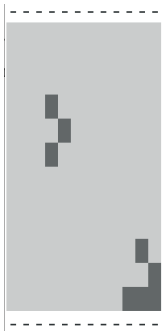


Figura 3. Visualizzazione sul terminale dell'automa cellulare.

## Esercizio 2

**Esecuzione del gioco: prima versione della procedura `main`.**

**Punti:** 10.

Scrivete un programma che, all'avvio, permetta all'utente di giocare al *Game of Life*. Il programma presenterà all'utente quattro opzioni:

1 visualizzare la griglia di gioco;
2 inserire una cella viva;
3 passare al prossimo stato dell'automa;
4 uscire dal programma.

Se l'utente seleziona l'opzione 1, allora il programma stampa a schermo il contenuto della griglia utilizzando la funzione `show()` sviluppata nell'esercizio precedente.

Se l'utente seleziona l'opzione 2, allora il programma chiede all'utente due interi $i$ e $j$ e fissa il valore di `aut[i][j]` a 1. Effettuate gli opportuni controlli sui valori inseriti.

Se l'utente seleziona l'opzione 3, allora il programma scansiona la matrice `aut` ed applica le regole del *Game of Life* spiegate nella premessa.

Per implementare tale opzione scrivete le seguenti funzioni:

- `int next_cell_state()` — questa funzione, di prototipo opportuno, avrà come parametri la matrice `aut` e due numeri interi $i$ e $j$, controllerà tutte le celle vicine alla cella $(i, j)$ e restituirà il nuovo valore della cella $(i, j)$ determinato dalle regole di cui sopra;
- `next_automata_state()` — questa funzione, di prototipo opportuno, riceverà in ingresso l'automa `aut` chiamerà la funzione `next_cell_state()` per ogni cella della matrice `aut`. I risultati di queste chiamate andranno scritti in una griglia di appoggio `new_aut` che, infine, andrà a sovrascrivere la matrice `aut`.

Per un esempio di esecuzione del programma si veda la Fig. 4.

## Esercizio 3

**Lettura/Scrittura da File.**

**Punti:** 9.

Si aggiungano al `main` due opzioni che permettano all'utente di leggere o scrivere un automa da/su file. In accordo con le specifiche dell'esercizio 1, assumete che il formato del file sia, ad esempio, il seguente:

```
0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0
```

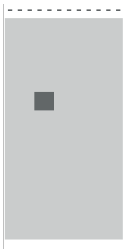
Potete usare come esempi i file `dati1.txt` e `dati2.txt`, allegati al tema d'esame. Implementate le seguenti funzioni:

- `int importa()` — questa funzione, di prototipo opportuno, riceverà in ingresso un puntatore a file e l'automa `aut`, leggerà il file puntato e salverà il contenuto nella matrice `aut`, restituendo infine il numero di celle lette;
- `int esporta()` — questa funzione, di prototipo opportuno, riceverà in ingresso un puntatore a file e l'automa `aut`, scriverà nel file puntato il contenuto della matrice `aut`, restituendo infine il numero di caratteri scritti.

Scrivete gli opportuni controlli ai valori restituiti dalle due funzioni, stampando a schermo messaggi d'errore o di successo appropriati.

```
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
Fai la tua scelta:2
Inserisci riga:3
Inserisci colonna:3
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
Fai la tua scelta:2
Inserisci riga:4
Inserisci colonna:4
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
Fai la tua scelta:2
Inserisci riga:5
Inserisci colonna:3
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
Fai la tua scelta:1
------------
```

```
------------
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
Fai la tua scelta:3
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
Fai la tua scelta:1
------------
```



```
------------
1.      mostra automa corrente.
2.      rendi viva una cella.
3.      prossimo stato.
4.      Esci.
```

FIGURA 4. Esempio di esecuzione del programma dell'Esercizio 2.

## Esercizio 4

**Sequenza di stati.**

**Punti:** 4.

Implementate una funzione `run()`, di prototipo opportuno, che riceverà in ingresso l'automa `aut`, chiederà all'utente un numero intero $n$ e produrrà una sequenza di $n$ stati visualizzando ogni transizione.

Aggiungere l'opzione `n stati` nel menù che permetta all'utente di richiamare tale funzione.

*Istruzioni per la consegna*

- Funzioni da consegnare (in un unico file sorgente).

  `main`

  `show`

  `next_cell_state`

  `next_automata_state`

  `esporta`

  `importa`

  `run`

  Funzioni di input/output su file

  Tutte le eventuali altre funzioni ausiliarie

  che abbiate scritto.

- Sito per la consegna:

  `https://upload.mat.unimi.it`

- Autenticarsi con le proprie credenziali di posta elettronica d'ateneo.
- Consegnare solo il codice sorgente (file `*.c`), non l'eseguibile.

(D. Valota) Dipartimento di Informatica *Giovanni Degli Antoni*, Università degli Studi di Milano, via Giovanni Celoria 18, I-20133 Milano
*E-mail address*: `valota@di.unimi.it`