



I VETTORI

Manuale linguaggio C

Vettori (array)

Un *array* è una **variabile aggregata** in grado di contenere un certo numero di dati tutti dello **stesso tipo**, cui è possibile accedere individualmente mediante il nome comune e referenziando lo specifico elemento tramite il suo **indice**.

IL C alloca in memoria gli elementi di un *array* in posizioni adiacenti, associando l'indirizzo più basso al primo elemento, e il più alto all'ultimo. Dichiarazione:

tipo NomeVariabile [dimensione];

dove *tipo* è il tipo degli elementi del vettore, *dimensione* è una costante che indica quanti elementi deve contenere il vettore. La dichiarazione serve al compilatore per riservare in memoria lo spazio sufficiente al vettore. Lo spazio occupato dal vettore sarà:

$\text{numero_byte} = \text{sizeof}(\text{tipo}) * \text{dimensione};$

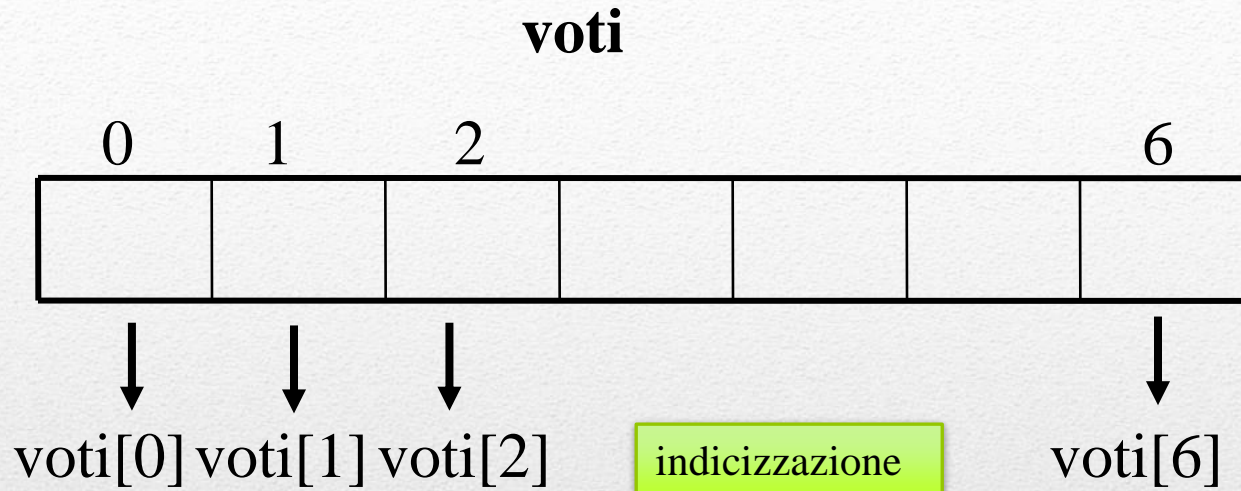
```
int voti[200];
```

```
#define N_VOTI 200;
```

```
int voti[N_VOTI];
```

```
//risulta più facile modificare la lunghezza del vettore
```

Vettori (array)



```
int voti[7];
```

Espressioni della forma `voti[i]` sono degli **lvalue** quindi possono essere usate come delle normali variabili:

```
voti[0] = 0;  
printf("%d\n", voti[5]);  
++voti[i];
```

Dichiarazione

Le istruzioni di dichiarazione di un array e quelle di indicizzazione sono molto simili nella forma, ma molto diverse nel significato

```
/* La seguente è una dichiarazione;  
 * il valore 365 specifica il numero  
 * di elementi dell'array  
 */  
int daily_temp[365];
```

```
/* I seguenti sono riferimenti a elementi  
 * dell'array; i valori 0,1,2,... specificano  
 * gli elementi a cui accedere  
 */  
daily_temp[0] = 2;  
daily_temp[1] = 5;  
daily_temp[2] = 3;  
... ..
```

Agli elementi di un vettore si accede spesso tramite costrutti iterativi, per effettuare la stessa operazione su ciascun elemento del vettore. Esempio:

```
for (i=0; i<N; i++)  
    voti[i]=0;                //azzerare gli elementi del vettore  
for (i=0; i<N; i++)  
    scanf("%d", &voti[i]);    //legge dati e li inserisce nel vettore  
for (i=0; i<N; i++)  
    sum += voti[i];           //somma gli elementi del vettore
```

Dichiarazione

Esempio: Calcolo della temperatura media annua

```
#include <stdio.h>
#include <stdlib.h>
#define DAYS_IN_YEAR 365

int main()
{
    int j, sum=0;
    int daily_temp[DAYS_IN_YEAR];

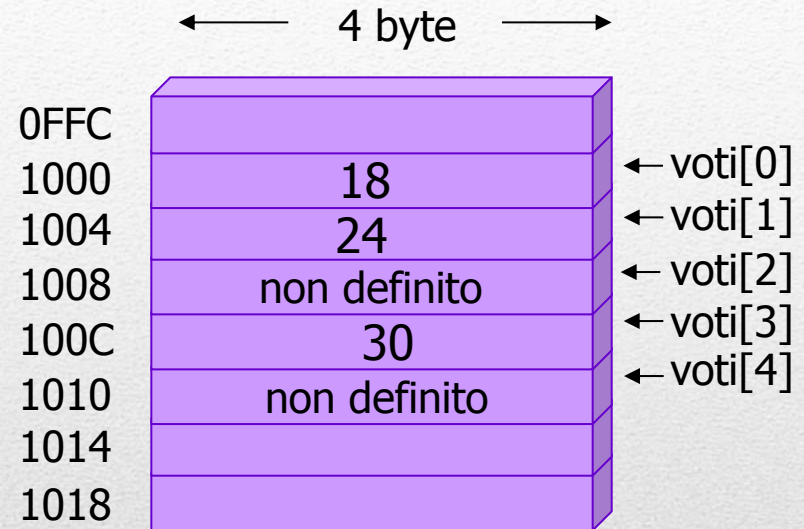
    /* inizializzazione dei valori di daily_temp[] */

    for (j=0; j<DAYS_IN_YEAR; j++)
        sum += daily_temp[j];
    printf("La temperatura media dell'anno è %f\n", sum/DAYS_IN_YEAR);
    return 0;
}
```

Memorizzazione dei vettori

Esempio

```
int voti[5]; /* dichiarazione */  
voti[0] = 18;  
voti[1] = 24;  
voti[3] = 30;
```



voti[2] e voti[4] sono indefiniti: il contenuto delle posizioni di memoria è quello rimasto da esecuzioni precedenti (**garbage**)

Memorizzazione dei vettori

```
int voti[15];
```

```
...  
voti[0] = 12;  
voti[1] = 25;  
for(i = 2; i < 15; i++)  
    voti[i] = i;  
for(i = 0; i < 15; i++)  
    printf("voti[%d] = %d\n", i, voti[i]);  
...
```

N.B. gli elementi di un *array in C* sono numerati a partire da 0 (e non da 1)

Che accade se si tenta di referenziare un elemento che non fa parte dell'*array*, ad esempio *voti[15]*? Il compilatore (tipicamente) mette a disposizione del programmatore gli strumenti per gestire la memoria, senza preoccuparsi di controllarne l'operato. Per il compilatore, *voti[15]* è semplicemente la word che si trova a 30 byte dall'indirizzo al quale l'*array* è memorizzato.

Questo provoca un comportamento **non definito** del programma

Attenzione: il compilatore non controlla il rispetto dei limiti del vettore mentre vi si accede in fase di run time

```
int voti[15];

...
voti[15]=28;          //SBAGLIATO
```

```
int voti[15];

...
voti[15]=28;          //SBAGLIATO
```

Logicamente *voti[15]* non esiste. **Fisicamente**, *voti[15]* fa riferimento alla porzione di memoria (di dimensione adeguata alla memorizzazione di un intero) immediatamente successiva a *voti[14]*. L'effetto dell'istruzione di assegnamento è quello di memorizzare il valore 28 alla locazione:



L'effetto dipende da ciò a cui era precedentemente destinata la cella ...

Memorizzazione dei vettori

Consideriamo le diverse possibilità. La locazione *voti*[15] può:

- memorizzare una variabile del programma
- memorizzare un'istruzione del programma
- non essere allocata per gli usi del programma.

Il primo caso provoca un'alterazione nel valore della variabile coinvolta difficilmente individuabile in fase di debugging (non c'è nessuna istruzione che la modifichi direttamente).

Il secondo caso può provocare l'interruzione da parte del sistema del programma per aver tentato di eseguire un'istruzione illegale.

Il risultato del terzo caso dipende dal sistema operativo che si sta utilizzando. Certi sistemi operativi (WIN 95/98, ad esempio) non controllano completamente gli accessi in memoria, e quindi il programma può danneggiare valori che appartengono ad altri programmi (anche al s. o. stesso). Altri s. o. (WIN NT/2000 o Unix) si accorgono che sta accadendo una violazione di memoria e terminano il programma che l'ha causata.

Inizializzazione dei vettori

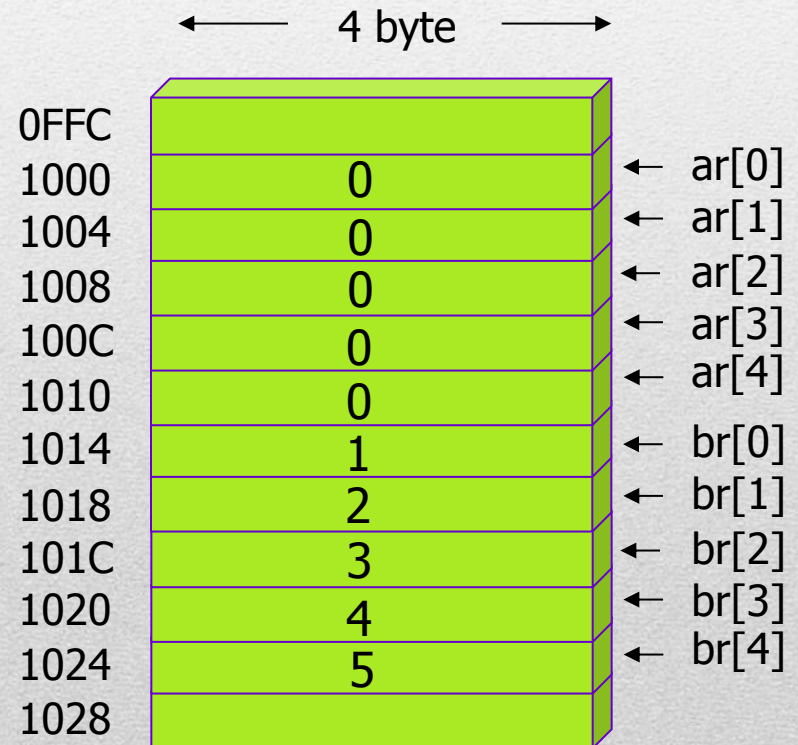
La presenza di valori indefiniti in alcuni elementi dell'array può provocare errori difficili da rilevare. Occorre quindi inizializzare l'intero vettore

- Singoli valori possono essere specificati, facendoli seguire alla dichiarazione dell'array, racchiusi fra parentesi graffe: tali valori devono essere espressioni costanti che possano essere convertite automaticamente nel tipo dell'array

Esempio

```
int ar[5]= {0,0,0,0,0};
```

```
int br[5] = {1,2,3,4,5};
```



Inizializzazione dei vettori

Specificare un numero maggiore di valori di inizializzazione, rispetto agli elementi dell'array, costituisce un errore segnalato dal compilatore

Specificare un numero minore di valori di inizializzazione, rispetto agli elementi dell'array, comporta l'inizializzazione a zero degli elementi rimanenti.

Esempio: La dichiarazione

```
int voti[5] = {18,22,30};
```

produce l'inizializzazione

```
{18,22,30,0,0}
```

Sfruttando questa caratteristica possiamo inizializzare a zero un intero vettore in modo molto semplice:

```
int voti[15]= {0}; // il valore iniziale è {0,0,0,0,0}
```

Inizializzazione dei vettori

Se vengono specificati i valori iniziali, può essere omessa la dimensione dell'array: il compilatore calcola automaticamente il numero degli elementi sulla base del numero dei valori iniziali specificati

Esempio:

```
char dr[] = {'a', 'b', 'c', 'd'};
```

comporta la creazione di un array di quattro elementi, di tipo char, caratterizzati dai valori iniziali

`dr[0] = 'a'`

`dr[1] = 'b'`

`dr[2] = 'c'`

`dr[3] = 'd'`

N.B.: Negli esempi riportati nei lucidi non viene controllato l'input inserito dall'utente per compattezza del codice e perché relativo agli specifici contesti in cui viene applicato. Ricordarsi sempre di farlo.

Progetto di programmazione – stampa reverse

Scrivere un programma C che chieda all'utente di inserire una serie di numeri, che li salvi in un vettore e poi li stampi in ordine inverso. Esempio:

```
#include <stdio.h>
#define N 10          //la si utilizzerà quattro volte all'interno del programma, molto utile

int main(void)
{
    int a[N], i;

    printf("Enter %d numbers: ", N);
    for (i = 0; i < N; i++)
        scanf("%d", &a[i]);

    printf("In reverse order:");
    for (i = N - 1; i >= 0; i--)
        printf(" %d", a[i]);
    printf("\n");

    return 0;
}
```

Esempio di uso dei vettori

```
/* Carica i punteggi di n concorrenti su due prove
   Determina la classifica */
#include <stdio.h>

#define MAX_CONC 1000    /* massimo numero di concorrenti */
#define MIN_PUN  1       /* punteggio minimo per ogni prova */
#define MAX_PUN  10      /* punteggio massimo per ogni prova */

main()
{
    float prova1[MAX_CONC], prova2[MAX_CONC], totale[MAX_CONC];
    int i, n;

    do {
        printf("\nNumero concorrenti: ");
        scanf("%d", &n);
    }
    while(n<1 || n>MAX_CONC);
    ...
}
```


Esempio di uso dei vettori

```
...
/* Per ogni concorrente, richiasta punteggio nelle due prove */
for(i=0; i<n; i++) {
    printf("\nConcorrente n.%d \n", i+1);
    do {
        printf("Prima prova: ");
        scanf("%f", &prova1[i]);
    }
    while(prova1[i]<MIN_PUN || prova1[i]>MAX_PUN);
    do {
        printf("Seconda prova: ");
        scanf("%f", &prova2[i]);
    }
    while(prova2[i]<MIN_PUN || prova2[i]>MAX_PUN);
}
/* Calcolo media per concorrente */
for(i=0; i<n; i++) totale[i] = (prova1[i]+prova2[i])/2;

printf("\n      CLASSIFICA\n");
for(i=0; i<n; i++)
    printf("%f  %f  %f \n", prova1[i], prova2[i], totale[i]);
}
```

Esempio di uso dei vettori

Numero concorrenti: **3**

Concorrente n.1

Prima prova: **8** Seconda prova: **7**

Concorrente n.2

Prima prova: **5** Seconda prova: **9**

Concorrente n.3

Prima prova: **8** Seconda prova: **8**

CLASSIFICA

8.000000 7.000000 7.500000

5.000000 9.000000 7.000000

8.000000 8.000000 8.000000

Dimensione logica e fisica

Un array è una collezione finita di N celle dello stesso tipo. Questo non significa che si debbano per forza usare sempre tutte.

La dimensione **logica** di un array può essere inferiore (mai superiore!) alla sua dimensione **fisica**

Spesso, la porzione di array realmente utilizzata dipende dai dati d'ingresso.

Esempio: memorizzare in un vettore i voti di una classe di studenti. Il vettore dovrà essere dimensionato al caso peggiore anche se poi in fase di esecuzione la porzione di esso effettivamente utilizzata potrà essere inferiore

Esempio: prodotto scalare

```
#include "stdio.h"
void main (void)
{
    int r;
    int vet1[10], vet2[10];
    int ps;
    for(r=0;r<10;r++){    //inizializzo vettore 1 di input
        printf("Inserisci elemento %d del vettore 1\n",r);
        scanf("%d",&vet1[r]);
    }
    for(r=0;r<10;r++){    //inizializzo vettore 2 di input
        printf("Inserisci elemento %d del vettore 2\n",r);
        scanf("%d",&vet2[r]);
    }
    //calcolo del prodotto scalare di due vettori
    ps=0;
    for(r=0;r<10;r++)
        ps+=vet1[r]*vet2[r];
    //stampa risultato
    printf("Il prodotto scalare è pari a %d\n",ps);
}
```

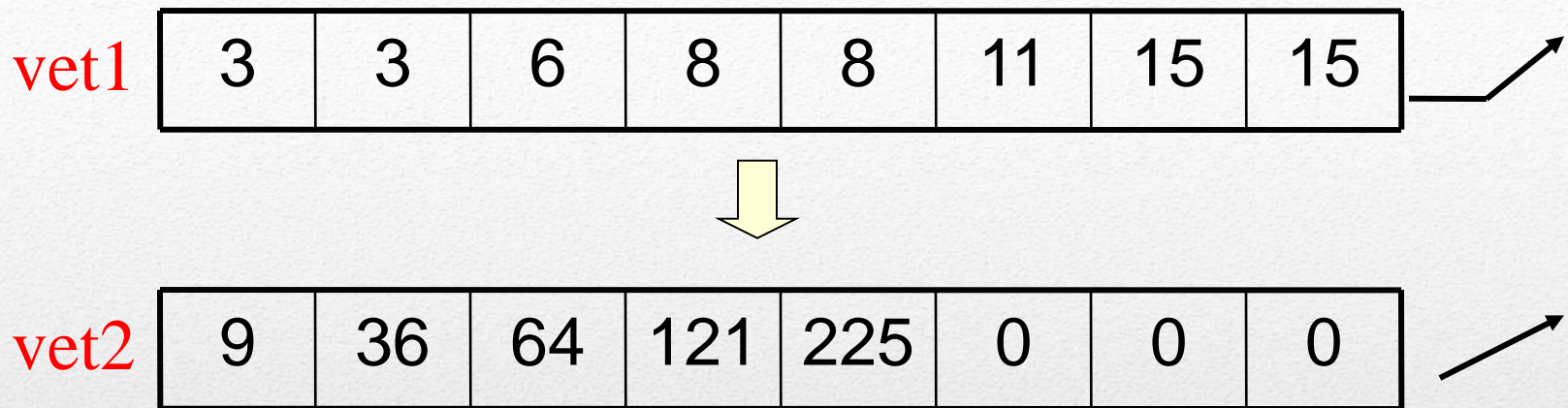

Esempio: reverse di un vettore

```
#include "stdio.h"
#define MAX_VET 100    //definisco dimensione massima del vettore

void main(void){
    int r,n_elem,mezzo,tmp;
    int vet[MAX_VET];

    printf("Inserisci numero di elementi del vettore (max 100) \n");
    scanf("%d",&n_elem); /*controllare input utente*/
    for(r=0;r<n_elem;r++){    /*inizializzo vettore di input*/
        printf("Inserisci elemento %d del vettore\n",r);
        scanf("%d",&vet[r]);
    }
    mezzo=(int)n_elem/2;    //calcolo quanti sono gli elementi da invertire
    for(r=0;r<mezzo;r++){
        tmp=vet[r];
        vet[r]=vet[n_elem-1-r];
        vet[n_elem-1-r]=tmp;
    }
    for(r=0;r<n_elem;r++)
        printf("Elemento posizione %d è %d\n",r,vet[r]);
}
```

Esempio: rottura di codice su vettore



Dato un vettore *vet1* di numeri interi in ordine non decrescente, costruire il vettore *vet2* contenente in ordine strettamente crescente i quadrati dei numeri contenuti in *vet1*.

Contesto generico: In un archivio, si vogliono elaborare, per ogni entità logicamente differente, certe informazioni. E' necessario eseguire l'elaborazione ogni volta che cambia il codice dell'entità (rottura di codice). Esempio: In un report, si vogliono presentare, per ogni studente, i voti conseguiti e la media relativa. E' necessario, in questo caso, eseguire la media dei voti ogni volta che cambia il codice dello studente.

Esempio: rottura di codice su vettore

```
#include "stdio.h"
#define MAX_VET 100

void main(void){

    int vet1[MAX_VET], vet2[MAX_VET];
    int r,i,pos_piena;

    printf("Inserisci numero elementi del vettore 1 (max 100) \n");
    scanf("%d",&r);          /*controllare input*/
    for(i=0;i<r;i++){
        printf("Inserisci elemento %d del vettore\n",i);
        scanf("%d",&vet1[i]);
    }
```

Esempio: rottura di codice su vettore

```
pos_piena=0;
i=0;
do{
while((vet1[i]==vet1[i+1]) && (i<r-1))
    i++;

vet2[pos_piena]=vet1[i]*vet1[i];
    pos_piena++;
i++;
}while(i<r);
for(i=pos_piena;i<r;i++)
    vet2[i]=0;
for(i=0;i<r;i++)
    printf("Elemento %d, del vettore: %d\n",i, vet2[i]);

}
```


Progetto di programmazione – calcolo interessi

Scrivere un programma C che stampi una tabella che illustra su un certo numero di anni il valore di un investimento di 100 dollari effettuato con diversi tassi di interesse. L'utente dovrà immettere il tasso di interesse e il numero di anni nei quali i soldi verranno investiti; la tabella dovrà mostrare il valore dell'investimento ogni anno a quel tasso e per i quattro tassi di interesse successivi. Esempio:

input: Enter interest rate: 6

Enter number of years: 5

output: vedi tabella

		fx		=D4+(D4*D3)			
	B	C	D	E	F	G	H
	Years	6%	7%	8%	9%	10%	
	1	106	107	108	109	110	
	2	112,36	114,49	116,64	118,81	121	
	3	119,1016	122,5043	125,9712	129,5029	133,1	
	4	126,2477	131,0796	136,0489	141,1582	146,41	
	5	133,8226	140,2552	146,9328	153,8624	161,051	

Nota: un ciclo iterativo (esempio for) ci aiuta per la stampa della prima riga.

Siccome i valori delle righe successive dipendono da quelli delle righe precedenti (vedi figura), possiamo memorizzare ogni riga in un vettore e basarci su questo per il calcolo dei valori delle righe successive

Progetto di programmazione – calcolo interessi

```
#include <stdio.h>
#define NUM_RATES ((int) (sizeof(value) / sizeof(value[0])))
#define INITIAL_BALANCE 100.00

int main(void)
{
    int i, low_rate, num_years, year;
    double value[5];

    printf("Enter interest rate: ");
    scanf("%d", &low_rate);
    printf("Enter number of years: ");
    scanf("%d", &num_years);

    printf("\nYears");
    for (i = 0; i < NUM_RATES; i++) {
        printf("%6d%%", low_rate + i);
        value[i] = INITIAL_BALANCE;
    }
    printf("\n");

    //CONTINUA
```


Progetto di programmazione – calcolo interessi

```
//CONTINUA
```

```
for (year = 1; year <= num_years; year++) {  
    printf("%3d  ", year);  
    for (i = 0; i < NUM_RATES; i++) {  
        value[i] += (low_rate + i) / 100.0 * value[i];  
        printf("%7.2f", value[i]);  
    }  
    printf("\n");  
}  
  
return 0;  
}
```

Vettori e indirizzi di memoria

Un *array*, come qualsiasi altro oggetto in memoria, ha un **indirizzo**, individuato e scelto dal compilatore. Il programmatore non può modificarlo, ma può conoscerlo attraverso il **nome dell'array** stesso, usandolo come un puntatore.

In C, il nome di un array equivale ad un puntatore all'area di memoria assegnata al vettore. Es.:

```
int *iPtr;  
int vettore[15]={10,20,30,40,50};  
printf("indirizzo di vettore: %X\n", vettore);  
iPtr = vettore;  
printf("indirizzo di vettore : %X\n",iPtr);  
printf("primo elemento di vettore : %d\n",* vettore);  
printf("secondo elemento di vettore : %d\n",*(vettore +1));  
++iPtr;  
printf("secondo elemento di vettore : %d\n",*iPtr);
```

Un puntatore è una variabile ma il nome di un array è costante. Quindi

- “ptr++” e “ptr=arr” sono espressioni valide, ma
- “arr++” e “arr=ptr” non sono valide.

L'indice di un elemento di un array ne esprime l'**offset** (distanza in termini di numero di elementi, dal primo elemento dell'array): il primo elemento ha offset 0 rispetto a se stesso; il secondo ha offset 1 rispetto al primo; il terzo ha offset 2, cioè dista 2 elementi dal primo...



I VETTORI MULTIDIMENSIONALI

Manuale linguaggio C

Vettori multidimensionali

Il vettore può avere un qualsiasi numero di dimensioni. È quindi possibile creare matrici e, più in generale, array a più dimensioni

```
int matrice[N][M];
```

- **N** indica il numero di righe
numerate da 0 a N-1

- **M** indica il numero di colonne
numerate da 0 a M-1

quante

5 colonne

4 righe

	0				4
0					
			★		
3					

Matrici

Per selezionare la cella di indici i, j :

$x = \text{matrice}[i][j];$

 $= \text{matrice}[1][2];$

Attenzione:

- $\text{matrice}[i,j]$ ha un altro significato!!

l'espressione i,j denota un solo numero (j), non la coppia di indici necessaria!

- $\text{matrice}[k]$ denota l'intera riga k
non c'è modo di denotare un'intera colonna

In C un array a 2 dimensioni è un array ad una dimensione in cui ognuno dei suoi elementi è un array

Matrici

Esempio:

```
int mat[4][3];
```

mat contiene 4 righe e 3 colonne per un totale di dodici elementi; per accedere a ciascuno di essi si utilizzano due indici: il primo specifica la riga, il secondo la colonna.

Gli indici variano rispettivamente tra 0 e $r - 1$ e tra 0 e $c - 1$, dove r e c sono il numero di righe e il numero di colonne.

```
mat[0][0] mat[0][1] mat[0][2]  
mat[1][0] mat[1][1] mat[1][2]  
mat[2][0] mat[2][1] mat[2][2]  
mat[3][0] mat[3][1] mat[3][2]
```

Allocazione della memoria

```
char m[3][4];
```

m, m[0], &m[0][0] →

m[1], &m[1][0] →

m[2], &m[2][0] →



← 1 byte

Vettori multidimensionali

Affrontare una dimensione alla volta:

```
int matrice [10][5];
```



/*vettore di 10 elementi

ogni elemento è un vettore di 5 interi*/

&matrice[i] /*indirizzo dell'i-esimo vettore di 5 elementi*/

&matrice[i][j] /*indirizzo dell'elemento i,j della matrice*/

```
char stringhe [10][30];
```

/*vettore di 10 stringhe da 30 caratteri*/

Una struttura dati per

- Le temperature di un mese:

`int M[31]` *e' un array di int (31 elementi)*

- Tutti i mesi dell' anno:

`int A[12] [31]` *e' un array di int _ [31] (12 elementi)*

- Le temperature del XX secolo:

`int S[100] [12][31]`
e' un array di int _[12][31] (100 elementi)

Qual e' il tipo di M[2]? A cosa corrisponde?

Qual e' il tipo di A[1]? " " ?

Qual e' il tipo di A[1][2]? " " ?

Matrici

Esempio: Visualizza gli elementi della matrice e calcola la somma

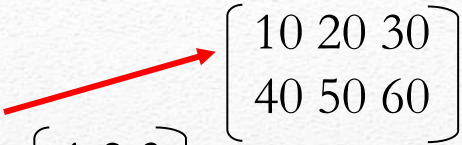
```
#include <stdio.h>
#define NROW 4
#define NCOL 3

int main(void)
{
    int m[NROW][NCOL] = { {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} };
    float somma = 0;
    int row,col;

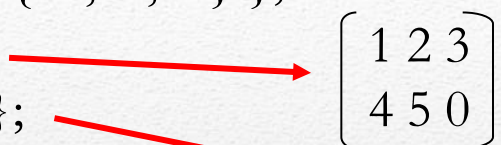
    for (row=0;row<NROW;row++) {
        for (col=0;col<NCOL;col++)
            printf("%5d", m[row][col]);
        printf("\n");
    }
    for (row=0;row<NROW;row++)
        for (col=0;col<NCOL;col++)
            somma += m[row][col];
    printf("La somma è %f\n", somma);
}
```


Inizializzazioni nelle dichiarazioni

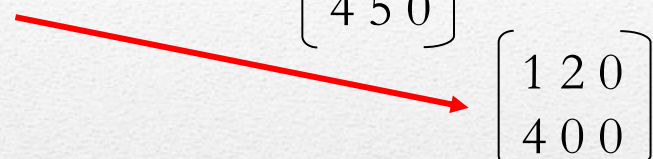
```
int vet1[2][3]= { {10,20,30}, {40,50,60} };
```


$$\begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{bmatrix}$$

```
int vet2[2][3]= { 1,2,3,4,5 };
```


$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \end{bmatrix}$$

```
int vet3[2][3]= { {1,2}, {4} };
```


$$\begin{bmatrix} 1 & 2 & 0 \\ 4 & 0 & 0 \end{bmatrix}$$

Si possono anche omettere le parentesi graffe interne; in tal caso, una volta che il compilatore avrà riempito una riga, comincerà a riempire la successiva. Aumenta però il rischio di commettere errori e diminuisce la leggibilità del codice.

Qualsiasi vettore (qualsiasi variabile) può essere reso costante dichiarandolo con la parola chiave **const**. Nel qual caso, qualsiasi tentativo di modifica degli elementi da parte del programma verrà segnalato come errore dal compilatore.

```
const int m[NROW][NCOL] = { {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} };
```

// contiene informazioni di riferimento che non vogliamo vengano modificate

Esempio: Prodotto di matrici

Date $\text{mat1}[N][P]$ e $\text{mat2}[P][M]$ la *matrice prodotto* è:

$$\text{pmat}[i][j] = \sum_{k=0}^{P-1} \text{mat1}[i][k] * \text{mat2}[k][j]$$

per $i=1..N$, $j=1..M$

Il numero di colonne della prima matrice (P) deve essere uguale al numero di righe della seconda. La matrice pmat è dunque costituita da N righe e M colonne.

Se consideriamo le matrici immesse nell'esempio del lucido seguente:

$$\begin{aligned} \text{pmat}[2][4] = & \text{mat1}[2][0] * \text{mat2}[0][4] + \\ & \text{mat1}[2][1] * \text{mat2}[1][4] + \\ & \text{mat1}[2][2] * \text{mat2}[2][4] \end{aligned}$$

ossia

$$\text{pmat}[2][4] = 5*3 + 2*4 + 0*5 = 23$$

Prodotto di matrici

PRIMA MATRICE

1	0	0
22	-6	3
5	2	0
11	4	7

SECONDA MATRICE

2	0	4	0	3
0	1	5	1	4
21	1	2	2	5

MATRICE PRODOTTO

2	0	4	0	3
107	-3	64	0	57
10	2	30	2	23
169	11	78	18	84

`pmat[2][4] =`

$$5*3 + 2*4 + 0*5 = 23$$

Prodotto di matrici

```
#include <stdio.h>
#define N 4
#define P 3
#define M 5
int mat1[N][P];          /* prima matrice */
int mat2[P][M];          /* seconda matrice */
int pmat[N][M];          /* matrice prodotto */
main()
{
    int i, j, k;
    printf("\n \n CARICAMENTO DELLA PRIMA MATRICE \n \n");
    for(i=0; i<N; i++)
        for(j=0; j<P; j++) {
            printf("Inserisci linea %d colonna %d val:", i, j);
            scanf("%d", &mat1[i][j]);
        };
    printf("\n \n CARICAMENTO DELLA SECONDA MATRICE \n \n");
    for(i=0; i<P; i++)
        for(j=0; j<M; j++) {
            printf("Inserisci linea %d colonna %d val:", i, j);
            scanf("%d", &mat2[i][j]);
        };
    ...
}
```

Prodotto di matrici

```
.../* Calcolo del prodotto */
for(i=0; i<N; i++)
    for(j=0; j<M; j++) {
        pmat[i][j] = 0;
        for(k=0; k<P; k++)
            pmat[i][j] = pmat[i][j] + mat1[i][k] * mat2[k][j];
    };
printf("\n \n PRIMA MATRICE \n ");
for(i=0; i<N; i++) {
    printf("\n");
    for(j=0; j<P; j++) printf("%5d", mat1[i][j]);
}
printf("\n \n SECONDA MATRICE \n ");
for(i=0; i<P; i++) {
    printf("\n");
    for(j=0; j<M; j++) printf("%5d", mat2[i][j]);
}
printf("\n \n MATRICE PRODOTTO \n ");
for(i=0; i<N; i++) {
    printf("\n");
    for(j=0; j<M; j++) printf("%5d", pmat[i][j]);
}
}
```

Prodotto di matrici

CARICAMENTO DELLA PRIMA MATRICE

```
Inserisci linea 0 colonna 0 val:1  
Inserisci linea 0 colonna 1 val:0  
Inserisci linea 0 colonna 2 val:0  
Inserisci linea 1 colonna 0 val:22  
Inserisci linea 1 colonna 1 val:-6  
Inserisci linea 1 colonna 2 val:3  
Inserisci linea 2 colonna 0 val:5  
Inserisci linea 2 colonna 1 val:2  
Inserisci linea 2 colonna 2 val:0  
Inserisci linea 3 colonna 0 val:11  
Inserisci linea 3 colonna 1 val:4  
Inserisci linea 3 colonna 2 val:7
```

Prodotto di matrici

CARICAMENTO DELLA SECONDA MATRICE

Inserisci	linea	0	colonna	0	val:	2
Inserisci	linea	0	colonna	1	val:	0
Inserisci	linea	0	colonna	2	val:	4
Inserisci	linea	0	colonna	3	val:	0
Inserisci	linea	0	colonna	4	val:	3
Inserisci	linea	1	colonna	0	val:	0
Inserisci	linea	1	colonna	1	val:	1
Inserisci	linea	1	colonna	2	val:	5
Inserisci	linea	1	colonna	3	val:	1
Inserisci	linea	1	colonna	4	val:	4
Inserisci	linea	2	colonna	0	val:	21
Inserisci	linea	2	colonna	1	val:	1
Inserisci	linea	2	colonna	2	val:	2
Inserisci	linea	2	colonna	3	val:	2
Inserisci	linea	2	colonna	4	val:	5

Progetto di programmazione – cifre ripetute

Scrivere un programma C che stampi una tabella che illustra per ogni cifra quante volte appare all'interno di un numero inserito dall'utente. Esempio:

input: Enter a number: 41271092

output:	Digit:	0	1	2	3	4	5	6	7	8	9
	Occurrences:	1	2	2	0	1	0	0	1	0	1

Progetto di programmazione – cifre ripetute

```
#include <stdio.h>

int main(void)
{
    int digit_count[10] = {0};
    int digit;
    long n;

    printf("Enter a number: ");
    scanf("%ld", &n);

    while (n > 0) {
        digit = n % 10;
        digit_count[digit]++;
        n /= 10;
    }

    // CONTINUA
```

Progetto di programmazione – cifre ripetute

```
// CONTINUA
```

```
printf ("Digit:   ");  
for (digit = 0; digit <= 9; digit++)  
    printf("%3d", digit);  
printf("\nOccurrences:");  
for (digit = 0; digit <= 9; digit++)  
    printf("%3d", digit_count[digit]);  
printf("\n");  
  
return 0;  
}
```


Numeri pseudo random

La funzione **rand()** consente di estrarre un numero pseudo-casuale ogni volta che viene richiamata. Per l'uso di tale funzione è richiesto lo header **<stdlib.h>**.

```
n=rand(); // nell'intervallo compreso tra 0 e RAND_MAX (costante definita in stdlib.h)
```

La sequenza di numeri ottenuta con la `rand()` è però sempre la stessa, per avere una sequenza che sia imprevedibile è necessario fornirle un seme di avvio diverso ogni volta che essa viene richiamata. Per tale scopo è necessario usare la funzione **srand()**. Il parametro che deve essere passato alla `srand()` è un *unsigned int*. Esempio:

```
srand(437U);
```

Ogni valore diverso del seme dà inizio ad una diversa sequenza di numeri. Si può usare il clock di macchina.

```
srand ((unsigned int) time((NULL)));
```

La funzione **time(NULL)** ritorna l'ora corrente. Il valore restituito da tale funzione viene convertito tramite un *cast* al tipo *unsigned int*.

Tali funzioni si trovano nella libreria «time.h» è perciò necessario includere lo statement.

Numeri pseudo random

```
#include "stdio.h"  
#include "stdlib.h"  
#include "time.h"
```

Scrivere un programma che generi un numero a caso e che chieda iterativamente all'utente di indovinarlo.

```
void main (void)  
{  
  
    int guess;  
    int magic;  
  
    srand((unsigned) time(NULL));  
    magic = rand();  
    do {  
        printf("Indovina il numero magico: ");  
        scanf("%d", &guess);  
        if (guess==magic)  
            printf("Bravo!!\n");  
        else printf("Sbagliato. Riprova.\n");  
    } while (magic != guess);  
}
```


Numeri pseudo random

```
#include "stdio.h"
#include "stdlib.h"
#include "time.h"

void main (void)
{

    int guess;
    int magic;

    srand((unsigned) time(NULL));
    magic = rand()%10;
    do {
        printf("Indovina il numero magico: ");
        scanf("%d", &guess);
        if (guess==magic)
            printf("Bravo!!\n");
        else printf("Sbagliato. Riprova.\n");
    } while (magic != guess);
}
```

... un po' più facile ...

Numeri pseudo random

```
#include "stdio.h"
#include "stdlib.h"
#include "time.h"

void main (void)
{
    int guess;
    int magic;
    srand((unsigned) time(NULL));
    printf("Inserisci primo estremo: \n");
    scanf("%d", &a);
    printf("Inserisci secondo estremo: \n");
    scanf("%d", &b);
    magic = rand() %(b-a)+a;
    do {
        printf("Indovina il numero magico: ");
        scanf("%d", &guess);
        if (guess==magic)
            printf("Bravo!!\n");
        else printf("Sbagliato. Riprova.\n");
    } while (magic != guess);
}
```


Esempio: somma matrici

```
#include "stdio.h"
void main (void)
{ int r,c;
  int mat1[10][5], mat2[10][5], mat3[10][5];
  for(r=0;r<10;r++)
    for(c=0;c<5;c++){
      printf("Inserisci elemento %d, %d della matrice 1\n",r,c);
      scanf("%d",&mat1[r][c]);          //inizializzo matrice 1 di input
    }
  for(r=0;r<10;r++)
    for(c=0;c<5;c++){
      printf("Inserisci elemento %d, %d della matrice 2\n",r,c);
      scanf("%d",&mat2[r][c]);          //inizializzo matrice 2 di input
    }
  for(r=0;r<10;r++)
    for(c=0;c<5;c++)
      mat3[r][c]=mat1[r][c]+mat2[r][c];    //calcolo della somma delle matrici
  for(r=0;r<10;r++)
    for(c=0;c<5;c++)
      printf("Elemento riga %d colonna %d è %d\n",r,c,mat3[r][c]);
}
```


Esempio: ricerca del massimo in matrice

```
#include "stdio.h"
#define MAX_MATR 100

void main(void) {
    int r,c,i,j,riga_max,colonna_max,max;
    int mat[MAX_MATR][MAX_MATR];

    printf("Inserisci numero di righe della matrice (max 100) \n");
    scanf("%d",&r);

    printf("Inserisci numero di colonne della matrice (max 100) \n");
    scanf("%d",&c);

    for(i=0;i<r;i++)
        for(j=0;j<c;j++) {
            printf("Inserisci elemento %d, %d della matrice\n",i,j);
            scanf("%d",&mat[i][j]);
        }
    // CONTINUA
```

Esempio: ricerca del massimo in matrice

```
// CONTINUA
```

```
max=0;                                /*ipotesi su valori della matrice*/
riga_max=colonna_max=1000;

for(i=0;i<r;i++)
    for(j=0;j<c;j++)
        if (mat[i][j]>max){
            max=mat[i][j];
            riga_max=i;
            colonna_max=j;
        }
printf("El. massimo: %d, in posizione %d, %d\n",max,riga_max,colonna_max);
}
```


Esempio: elaborazione di matrici

mat1 →	5	6		30	5	← mat2
	32	8		256	8	
	2	5		10	2	
	1	9		9	1	
	3	4		12	3	
	7	2		14	2	
	A	B		X	Y	

$$X = A * B$$

$$Y = \min(A, B)$$

Esempio: elaborazione di matrici

```
#include "stdio.h"
#define MAX_MATR 100
void main(void) {
    int r,i,j;
    int mat1[MAX_MATR][2], mat2[MAX_MATR][2];

    printf("Inserisci numero di righe delle matrici (max 100) \n");
    scanf("%d",&r);

    for(i=0;i<r;i++)
        for(j=0;j<2;j++) {
            printf("Inserisci elemento %d, %d della matrice\n",i,j);
            scanf("%d",&mat1[i][j]);
        }
    for(i=0;i<r;i++) {
        mat2[i][0]=mat1[i][0]*mat1[i][1];
        mat2[i][1]=mat1[i][0]<mat1[i][1] ? mat1[i][0] : mat1[i][1];
    }
    for(i=0;i<r;i++)
        printf("Riga %d, prodotto = %d, minimo = %d \n",i, mat2[i][0], mat2[i][1]);
}
```


Esempio: rottura di codice su matrice

mat1	3	15	34	mat2	3	83	34	
	3	3	2		7	23	9	$Q = \text{somma degli } N$
	3	65	3		8	8	61	$\text{a parità di } M$
	7	23	9		9	40	20	$R = \text{max dei } P$
	8	8	61		0	0	0	$\text{a parità di } M$
	9	19	7		0	0	0	
	9	21	20		0	0	0	
	M	N	P		M	Q	R	

Sia data la matrice *mat1* con valori non decrescenti della prima colonna.
Costruire *mat2* con i valori della prima colonna in ordine strettamente crescente,
la somma e il massimo dei corrispondenti nella seconda e terza colonna.

Esempio: rottura di codice su matrice

```
#include "stdio.h"
#define MAX_MATR 100

void main(void) {

    int r,i,j,somma,max,pos_piena;
    int mat1[MAX_MATR][3], mat2[MAX_MATR][3];

    printf("Inserisci numero di righe della matrice 1 (max 100) \n");
    scanf("%d",&r);

    for(i=0;i<r;i++)
        for(j=0;j<3;j++){
            printf("Inserisci elemento %d, %d della matrice\n",i,j);
            scanf("%d",&mat1[i][j]);
        }
    // CONTINUA
```

Esempio: rottura di codice su matrice

```
// CONTINUA
```

```
pos_piena=0;
```

```
i=0;
```

```
do{      max=0;somma=0;
```

```
    while((mat1[i][0]==mat1[i+1][0]) && (i<r-1)) {
```

```
        if(mat1[i][2]>max) max=mat1[i][2];
```

```
        somma+=mat1[i][1];
```

```
        i++;
```

```
    }
```

```
    if(mat1[i][2]>max) max=mat1[i][2];
```

```
    somma+=mat1[i][1];
```

```
    mat2[pos_piena][0]=mat1[i][0];
```

```
    mat2[pos_piena][1]=somma;
```

```
    mat2[pos_piena][2]=max;
```

```
    i++;
```

```
    pos_piena++;
```

```
}while(i<r);
```

```
for(i=pos_piena;i<r;i++)
```

```
    mat2[i][0]=mat2[i][1]=mat2[i][2]=0;
```

```
for(i=0;i<r;i++)
```

```
    for(j=0;j<3;j++)
```

```
        printf("Elemento %d, %d della matrice: %d\n",i,j,mat2[i][j]);
```

```
}
```

Progetto di programmazione – interessi mensili

Modificare il programma C relativo al calcolo degli interessi in modo da calcolare gli interessi composti *mensili* invece che *annuali*. Il formato dell'output non deve cambiare, il bilancio deve essere visibile ancora in intervalli annuali..

Esempio:



Progetto di programmazione – interessi mensili

```
#include <stdio.h>

#define NUM_RATES ((int) (sizeof(value) / sizeof(value[0])))
#define INITIAL_BALANCE 100.00

int main(void)
{
    int i, low_rate, month, num_years, year;
    double value[5];

    printf("Enter interest rate: ");
    scanf("%d", &low_rate);
    printf("Enter number of years: ");
    scanf("%d", &num_years);

    // CONTINUA
```

Progetto di programmazione – interessi mensili

```
// CONTINUA
```

```
printf("\nYears");
for (i = 0; i < NUM_RATES; i++) {
    printf("%6d%%", low_rate + i);
    value[i] = INITIAL_BALANCE;
}
printf("\n");

for (year = 1; year <= num_years; year++) {
    printf("%3d  ", year);
    for (i = 0; i < NUM_RATES; i++) {
        for (month = 1; month <= 12; month++)
            value[i] += ((double) (low_rate + i) / 12) / 100.0 * value[i];
        printf("%7.2f", value[i]);
    }
    printf("\n");
}

return 0;
}
```


Progetto di programmazione – studenti

Scrivere un programma C che stampi il punteggio ottenuto da cinque studenti in cinque quiz. Il programma deve poi calcolare il punteggio totale e quello medio per ogni studente. Inoltre, andranno calcolati il punteggio medio, quello massimo e quello minimo per ogni quiz.

Progetto di programmazione – studenti

```
#include <stdio.h>

#define NUM_QUIZZES 5
#define NUM_STUDENTS 5

int main(void)
{
    int grades[NUM_STUDENTS][NUM_QUIZZES];
    int high, low, quiz, student, total;

    for (student = 0; student < NUM_STUDENTS; student++) {
        printf("Enter grades for student %d: ", student + 1);
        for (quiz = 0; quiz < NUM_QUIZZES; quiz++)
            scanf("%d", &grades[student][quiz]);
    }
    // CONTINUA
```


Progetto di programmazione – studenti

```
// CONTINUA
```

```
printf("\nStudent Total Average\n");  
for (student = 0; student < NUM_STUDENTS; student++) {  
    printf("%4d    ", student + 1);  
    total = 0;  
    for (quiz = 0; quiz < NUM_QUIZZES; quiz++)  
        total += grades[student][quiz];  
    printf("%3d    %3d\n", total, total / NUM_QUIZZES);  
}
```

```
// CONTINUA
```

Progetto di programmazione – studenti

```
// CONTINUA
```

```
printf("\nQuiz Average High Low\n");
for (quiz = 0; quiz < NUM_QUIZZES; quiz++) {
    printf("%3d    ", quiz + 1);
    total = 0;
    high = 0;
    low = 100;
    for (student = 0; student < NUM_STUDENTS; student++) {
        total += grades[student][quiz];
        if (grades[student][quiz] > high)
            high = grades[student][quiz];
        if (grades[student][quiz] < low)
            low = grades[student][quiz];
    }
    printf("%3d    %3d    %3d\n", total / NUM_STUDENTS, high, low);
}

return 0;
}
```

Progetto di programmazione – mano di carte

Scrivere un programma C che distribuisca una mano di carte scelte a caso da un mazzo da gioco standard. Il numero di carte assegnate deve essere inserito dall'utente. Esempio:

input: Enter numbers of cards in hand: 5

output: Your hand: 7c 2s 5d as 2h

Nota: estrarre a caso le carte e accertarsi di non scegliere due volte la stessa carta.

Progetto di programmazione – mano di carte

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define FALSE 0
#define TRUE 1

#define NUM_SUITS 4
#define NUM_RANKS 13

int main(void)
{
    bool in_hand[NUM_SUITS][NUM_RANKS] = {FALSE};
    int num_cards, rank, suit;
    const char rank_code[] = {'2','3','4','5','6','7','8','9','t','j','q','k','a'};    //per la stampa
    const char suit_code[] = {'c','d','h','s'};    //per la stampa

    srand((unsigned) time(NULL));

    // CONTINUA
```

Progetto di programmazione – mano di carte

```
// CONTINUA
```

```
printf("Enter number of cards in hand: ");
scanf("%d", &num_cards);

printf("Your hand:");
while (num_cards > 0) {
    suit = rand() % NUM_SUITS;    /* picks a random suit */
    rank = rand() % NUM_RANKS;   /* picks a random rank */
    if (!in_hand[suit][rank]) {
        in_hand[suit][rank] = TRUE;
        num_cards--;
        printf(" %c%c", rank_code[rank], suit_code[suit]);
    }
}
printf("\n");

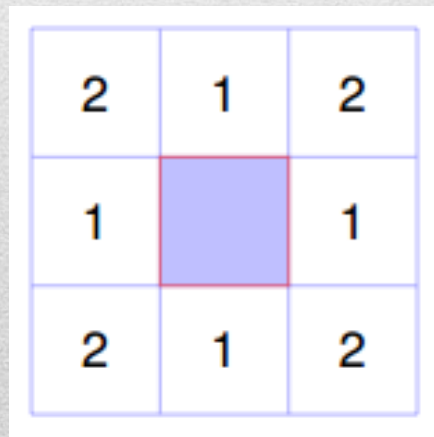
return 0;
}
```

Esempio: gioco della vita

Nel 1970 Martin Gardner pubblicò le regole di un nuovo gioco inventato dal matematico inglese John Horton Conway (*Scientific American* 223, (1970), pag 120-123)

Il solitario venne chiamato “The game of LIFE” grazie alle analogie con la nascita, evoluzione e morte di un gruppo organismi viventi.

Il gioco è una estensione agli automi cellulari in due dimensioni. Si effettua su una scacchiera ed ogni cellula ha 8 celle vicine (confinanti): 4 adiacenti ortogonali (1) e 4 sulle diagonali (2).



Esempio: gioco della vita

Una mappa di dimensione $N \times M$ rappresenta il mondo.

Ogni cella può essere occupata o meno da un organismo. Partendo da una configurazione iniziale di organismi, questa popolazione evolve nel tempo secondo tre regole genetiche:

- un organismo sopravvive fino alla generazione successiva se ha 2 o 3 vicini;
- un organismo muore, lasciando la cella vuota, se ha più di 3 o meno di 2 vicini;
- ogni cella vuota con 3 vicini diventa una cella di nascita e alla generazione successiva viene occupata da un organismo.

Tutti i cambiamenti di stato (nascita/morte) avvengono simultaneamente. Si visualizzi l'evoluzione della popolazione nel tempo.

Il concetto di “vicinanza” in una tabella raffigurante il mondo può essere interpretato in 2 modi:

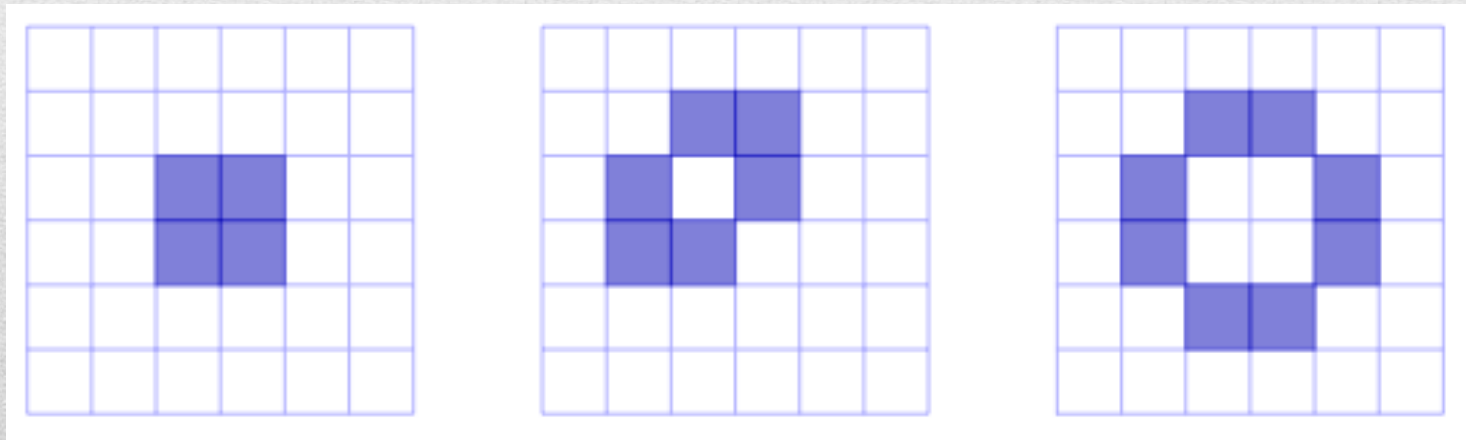
- Al di là dei bordi c'è il vuoto che non influenza il gioco, per cui ci sono punti interni che hanno 8 potenziali “vicini”, punti sulle righe e colonne estreme che hanno 5 “vicini”, punti ai vertici che hanno 3 “vicini”
 - I bordi estremi confinano tra di loro: la colonna “0” è “vicina” alla colonna “M”, così come la riga “0” è “vicina” alla riga “N” (con attenzione a trattare i vertici!)
-

Esempio: gioco della vita

L'interesse che suscitò il gioco nasce dalla scoperta di forme con schemi evolutivi particolari:

- forme statiche;
- oscillatori (forme periodiche);
- gliders (alianti), (oscillatori che si spostano nello spazio).

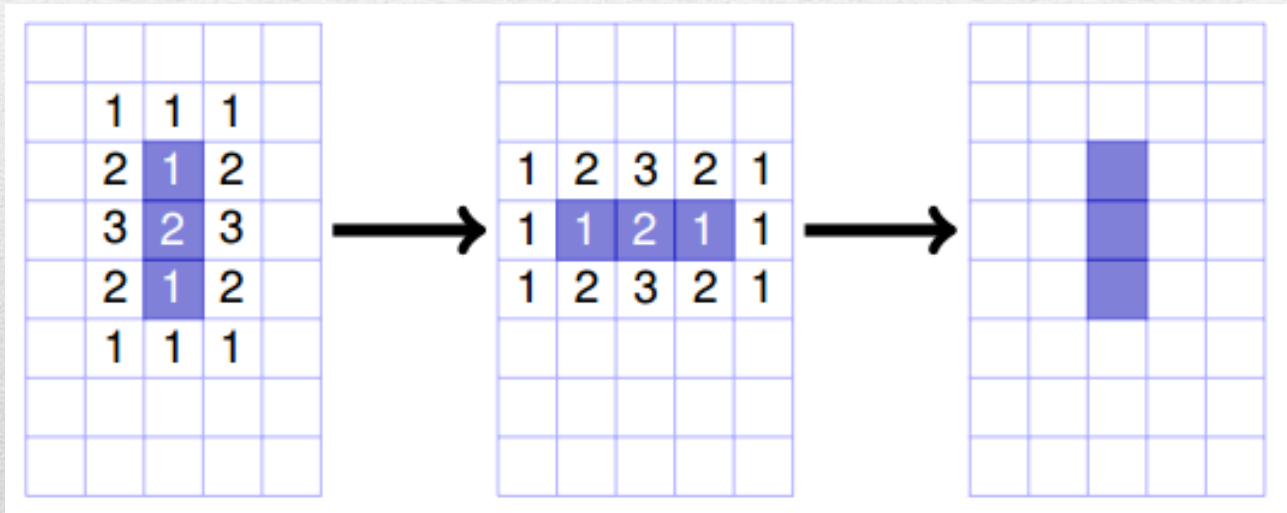
Esempio: schemi statici ("Still Life"):



Esempio: schemi oscillanti:

Gli oscillatori sono delle forme che si ripetono con un periodo $T > 1$

Le forme stazionarie hanno un periodo $T = 1$



Esempio: gioco della vita

Esempio: schemi glider:

I "Gliders" (alianti) sono degli oscillatori che si spostano lungo la griglia durante l'evoluzione del sistema

