ARRAY DI PUNTATORI

- Non ci sono vincoli sul tipo degli elementi di un vettore
- Possiamo dunque avere anche ***vettori di puntatori***

Ad esempio:

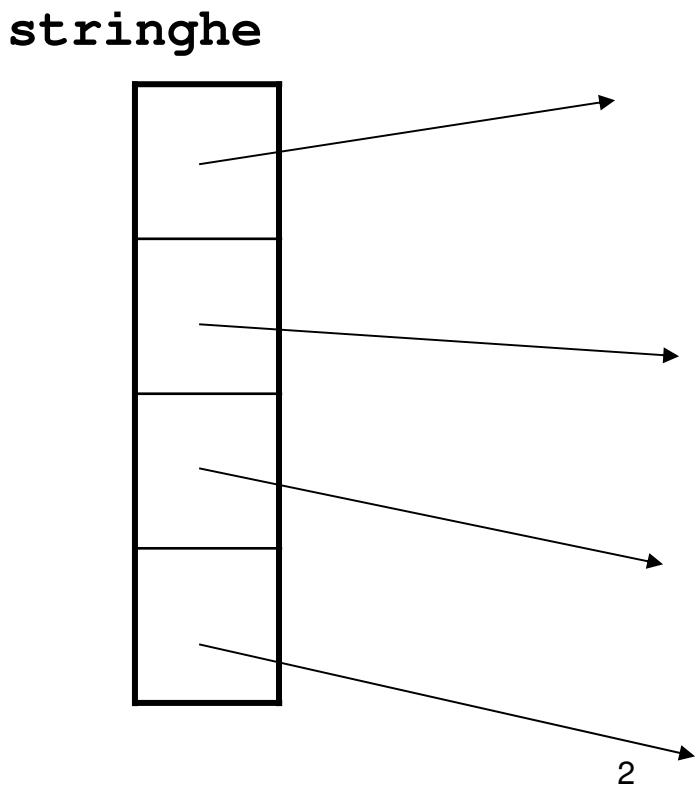
```
char * stringhe[4];
```

definisce un vettore di 4 puntatori a carattere (allocata memoria per 4 puntatori)

ARRAY DI PUNTATORI

`stringhe` sarà dunque una struttura dati rappresentabile nel modo seguente

I vari ***puntatori*** sono ***memorizzati in celle contigue***. Possono invece ***non essere contigue le celle che loro puntano***



INIZIALIZZAZIONE

Come usuale, anche gli array di puntatori a stringhe possono essere ***inizializzati***

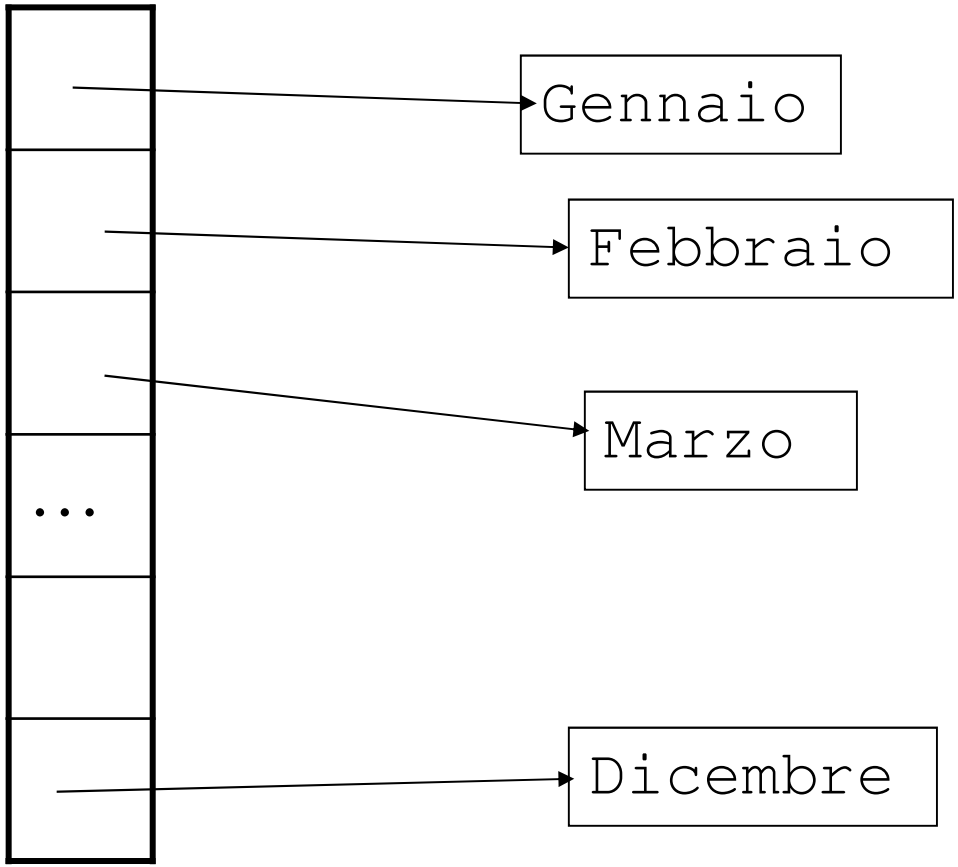
```
char * mesi[] =          {"Gennaio","Febbraio",
      "Marzo","Aprile","Maggio","Giugno",
      "Luglio","Agosto", "Settembre","Ottobre",
      "Novembre","Dicembre"};
```

I caratteri dell'i-esima stringa vengono posti in una locazione qualsiasi e ***in mesi[i] viene memorizzato un puntatore a tale locazione***

Come sempre, poiché l'ampiezza del vettore non è stata specificata, il compilatore conta i valori di inizializzazione e dimensiona il vettore di conseguenza

INIZIALIZZAZIONE

mesi



ARRAY MULTIDIMENSIONALI

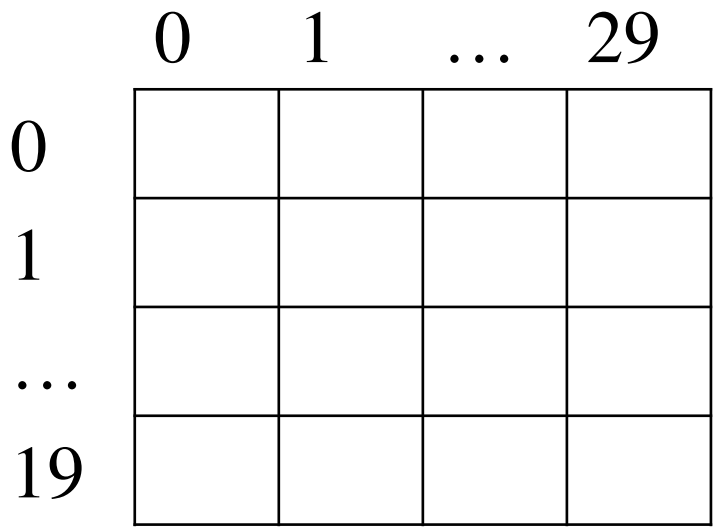
È possibile definire variabili di tipo array con più di una dimensione

```
<tipo> <identificatore>[dim1][dim2]…[dimn]
```

Array con due dimensioni vengono solitamente detti ***matrici***

```
float M[20][30];
```

Come sempre, ***allocazione statica***:
allocazione di 20x30 celle
atte a mantenere float



MATRICI

Per accedere all'elemento che si trova nella ***riga i*** e nella ***colonna j*** si utilizza la notazione

```
M[i][j]
```

Anche possibilità di vettori con più di 2 indici:

```
int C[20][30][40];
int Q[20][30][40][40];
```

MEMORIZZAZIONE

Le matrici multidimensionali sono ***memorizzate per righe in celle contigue***. Nel caso di M:

M[0][0]	M[0][1]	…	M[0][29]	M[1][0]	M[1][1]	…	M[1][29]	….	M[19][0]	M[19][1]	…	M[19][29]

E analogamente nel caso di più di 2 dimensioni: viene fatto variare prima l'indice più a destra, poi il penultimo a destra, e così via fino a quello più a sinistra

Per calcolare ***l'offset*** della cella di memoria dell'elemento (i,j) in una matrice bidimensionale (rispetto all'indirizzo di memorizzazione della prima cella – *indirizzo dell'array*):

LungRiga * i + j

Nel caso di M: **`M[i][j]`** elemento che si trova nella cella 30*i+j dall'inizio della matrice

Elemento: *(&M[0][0] + 30*i+j)

MEMORIZZAZIONE

In generale, se

<tipo> mat[dim$_1$][dim$_2$]…[dim$_n$]

mat[i$_1$][i$_2$]…[i$_{n-1}$][i$_n$]

è l'elemento che si trova nella cella $i_1*dim_2*\ldots*dim_n+\ldots + i_{n-1}*dim_n+i_n$ a partire dall'inizio del vettore

MATRICI

Si possono anche dichiarare dei tipi vettore multidimensionale

```
typedef <tipo> <ident>[dim1][dim2]…[dimn]
```

```
typedef  float MatReali [20] [30];
MatReali Mat;
```

è equivalente a

```
typedef float VetReali[30];
typedef VetReali MatReali[20];
MatReali Mat;
```

INIZIALIZZAZIONE

Come al solito, i vettori multidimensionali possono essere inizializzati con una lista di valori di inizializzazione racchiusi tra parentesi graffe

```
int matrix[4][4]={{1,0,0,0},{0,1,0,0},
                  {0,0,1,0}, {0,0,0,1}}
```

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

Puntatori e Vettori MULTIDIMENSIONALI

Anche un vettore multidimensionale è ***implementato in C come un puntatore all'area di memoria*** da cui partono le ***celle contigue*** contenenti il vettore

```
int a[10][4];
```

a vettore di 10 elementi, ciascuno dei quali è un vettore a 4 interi
tipo = "puntatore a vettore di 4 interi" non "puntatore a intero"

```
int ** b; int * c;
b=a;
c=a;
```
compila con warning

PUNTATORI E VETTORI MULTIDIMENSIONALI

Date le due definizioni

```
int a[10][4]; int *d[10];
```

la prima alloca 40 celle di ampiezza pari alla dim di un int mentre la seconda alloca 10 celle per contenere 10 puntatori a int

Uno dei vantaggi offerti dall'uso di vettori di puntatori consiste nel fatto che si possono realizzare ***righe di lunghezza variabile***

PUNTATORI E VETTORI MULTIDIMENSIONALI

```
char * mesi[] =     {"Gennaio","Febbraio",
     "Marzo","Aprile","Maggio","Giugno",
     "Luglio","Agosto", "Settembre",
     "Ottobre", "Novembre", "Dicembre"};
```

vengono create ***righe di lunghezza variabile***

Esempio: PRODOTTO MATRICI QUADRATE

Programma per il **prodotto (righe x colonne) di matrici quadrate NxN** a valori interi:

$$C[i,j] = \Sigma_{(k=1..N)} A[i][k]*B[k][j]$$

```
#include <stdio.h>
#define N 2
typedef int Matrici[N][N];

int main(){
int Somma,i,j,k;
Matrici A,B,C;
/* inizializzazione di A e B */
  for (i=0; i<N; i++)
      for (j=0; j<N; j++)
            scanf("%d",&A[i][j]);
  for (i=0; i<N; i++)
      for (j=0; j<N; j++)
            scanf("%d",&B[i][j]);
```

Esempio: PRODOTTO MATRICI QUADRATE

```
/* prodotto matriciale */
  for (i=0; i<N; i++)
      for (j=0; j<N; j++){
            Somma=0;
            for (k=0; k<N; k++)
                  Somma=Somma+A[i][k]*B[k][j];
            C[i][j]=Somma;}

/* stampa */
  for (i=0; i<N; i++) {
      for (j=0; j<N; j++)
            printf("%d\t",C[i][j]);
            printf("\n"); }
}
```

PASSAGGIO DI PARAMETRI

Nel caso di passaggio come parametro di un vettore bidimensionale a una funzione, nel prototipo della funzione ***va dichiarato necessariamente il numero delle colonne*** (ovvero la dimensione della riga)

Ciò è essenziale perché il compilatore sappia come accedere al vettore in memoria

PASSAGGIO DI PARAMETRI

Esempio: se si vuole passare alla funzione **`f()`** la matrice **`par`** occorre scrivere all'atto della definizione della funzione**:**

```
f(float par[20][30],…)
```

oppure

```
f(float par[][30],…)
```

perché il numero di righe è irrilevante ai fini dell'aritmetica dei puntatori su **`par`**

In generale, come già detto, ***soltanto la prima dimensione di un vettore multidimensionale può non essere specificata***

Esempio: PRODOTTO MATRICI QUADRATE

```
#include <stdio.h>
#define N 2
typedef int Matrici[N][N];

void prodottoMatrici(int X[][N], int Y[][N],
                     int Z[][N]) {
int Somma,i,j,k;
for (i=0; i<N; i++)
      for (j=0; j<N; j++){
            Somma=0;
            for (k=0; k<N; k++)
                  Somma=Somma+X[i][k]*Y[k][j];
            Z[i][j]=Somma;}
}
```

Esempio: PRODOTTO MATRICI QUADRATE

```
int main(void){
int Somma,i,j,k;
Matrici A,B,C;

for (i=0; i<N; i++) /* inizializ. di A e B */
      for (j=0; j<N; j++)
            scanf("%d",&A[i][j]);
for (i=0; i<N; i++)
      for (j=0; j<N; j++)
            scanf("%d",&B[i][j]);

prodottoMatrici(A,B,C); //in C verrà caricato il
  risultato del prodotto

for (i=0; i<N; i++) {          /* stampa */
      for (j=0; j<N; j++)
            printf("%d\t",C[i][j]);
            printf("\n"); }}
```