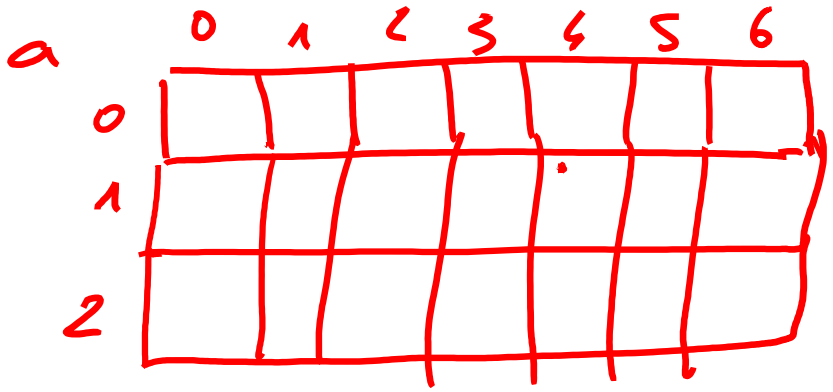


MATRICI

Vettori multidimensionali

ARRAY E TYPEDEF

A volte è utile definire un nuovo tipo di dato come array.
Si usa la solita sintassi del linguaggio C

```
typedef <vecchiotipo> <nuovotipo>;
```

Es `typedef char stringa[10];`

Char s[10];

Tipo

dichiara che il tipo stringa è un array di 10 caratteri.

A questo punto, posso definire variabili di questo tipo e usarle come normali array:

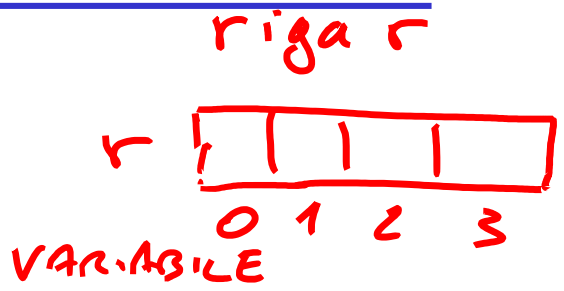
```
main()
{ stringa s;

  strcpy(s,"hello");

  printf("%c",s[1]);

}
```

Char s[10];

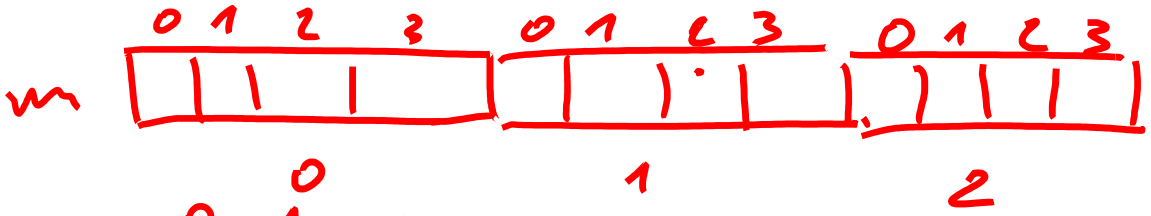
ARRAY E TYPEDEF

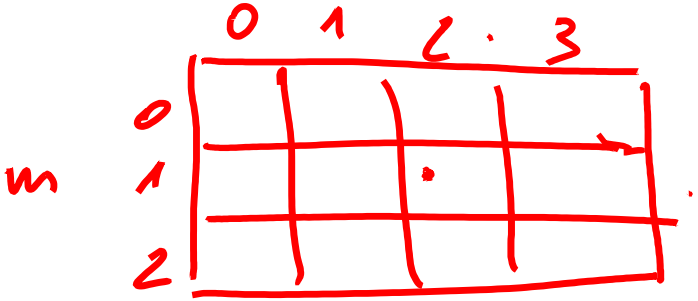
Si possono anche definire array di array:

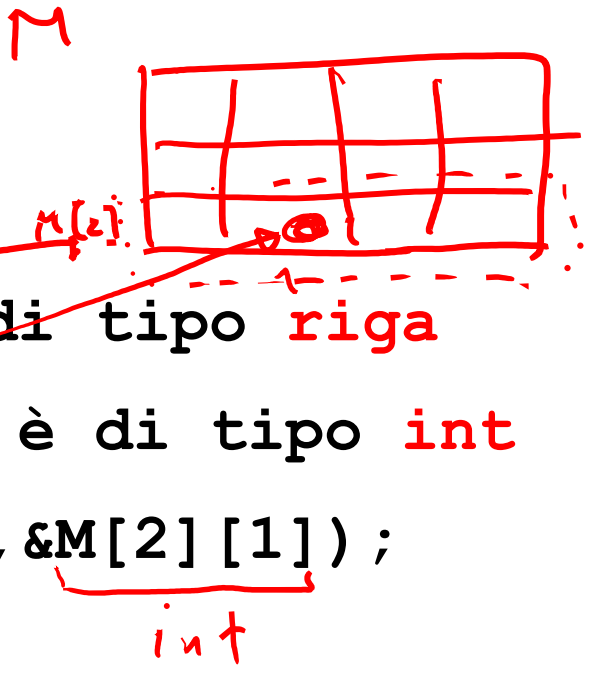
Es.
```
typedef int riga[4];
typedef riga matrice[3];
```



riga m[3]; m[1][2]








```
main()
{ matrice M;
  // M[2] è di tipo riga
  // M[2][1] è di tipo int
  scanf("%d",&M[2][1]);
}
```

S + M[2][1]

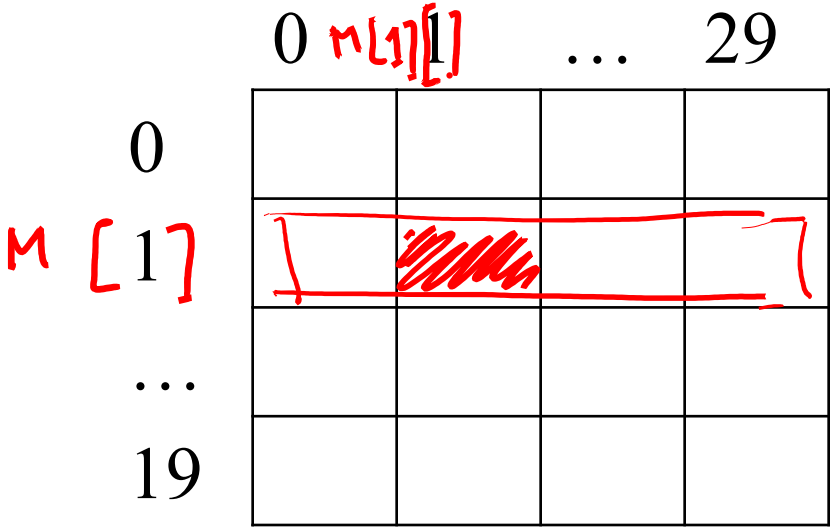
ARRAY MULTIDIMENSIONALI

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];
```



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];
```

MATRICI

Le matrici vengono memorizzate per righe

0 1 2 3 4 5 6 7 8 9 10 11

1000 1004 1008 1012 1016 1020 1024

```
int a[3][4];
```

4 byte

a[1][2]

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

FORTRAN

Quindi, per calcolare l'indirizzo della cella `a[i][j]`:

- parto dall'indirizzo della cella `a[0][0]` 1000
- aggiungo `i` moltiplicato per la lunghezza di una riga 1 · 4 · 4 = 16 1016
- sommo `j` 2 · 4 = 8 1024

MEMORIZZAZIONE

In generale, se

`<tipo> mat[dim1][dim2]…[dimn]`

int a 2 4 5

`mat[i1][i2]…[in-1][in]`

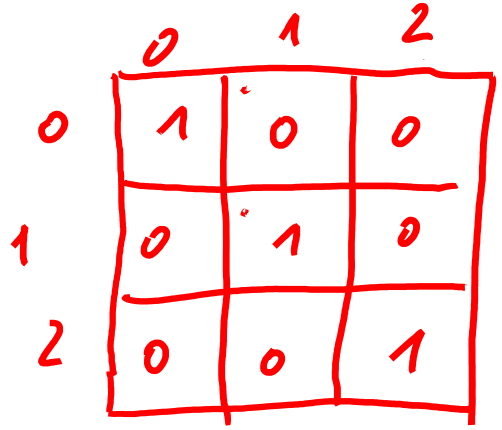
è l'elemento che si trova nella cella

indirizzo iniziale + ($i_1*dim_2*\ldots*dim_n+\ldots + i_{n-1}*dim_n+i_n$) · sizeof(tipo)

a partire dall'inizio del vettore

ESERCIZIO

Si inizializzi la matrice identità 3x3




```
int M[3][3];
// M[1][2] → 0

for (i=0; i<dim; i++)
  ... a[i] ..
```

```
for (i=0; i<d1; i++) {
  for (j=0; j<d2; j++) {
    ... M[i][j] ...
  }
}
```

int a[3]= {1,2,3};

INIZIALIZZAZIONE

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

{{1,0,0,0} ,{0,1,0,0} ,{0,0,1,0} {0,0,0,1}}

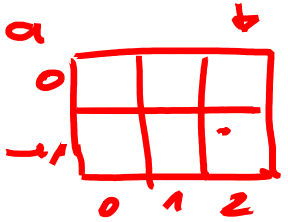
```
int matrix[4][4]=
  {{1,0,0,0},
   {0,1,0,0},
   {0,0,1,0},
   {0,0,0,1}};
```

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

PASSAGGIO DI PARAMETRI

Per passare una matrice come parametro ad una funzione, si usa la normale sintassi del C

```
char funzione(char a[2][3])
{ return a[1][2];
}


main()
{  char x,M[2][3];
   x = funzione(M);
}
```



Nella definizione devo dichiarare il tipo

Nell'invocazione metto solo il nome della matrice, senza quadre

PASSAGGIO DI PARAMETRI

Siccome la matrice è un array, alla funzione viene passato solo l'indirizzo iniziale

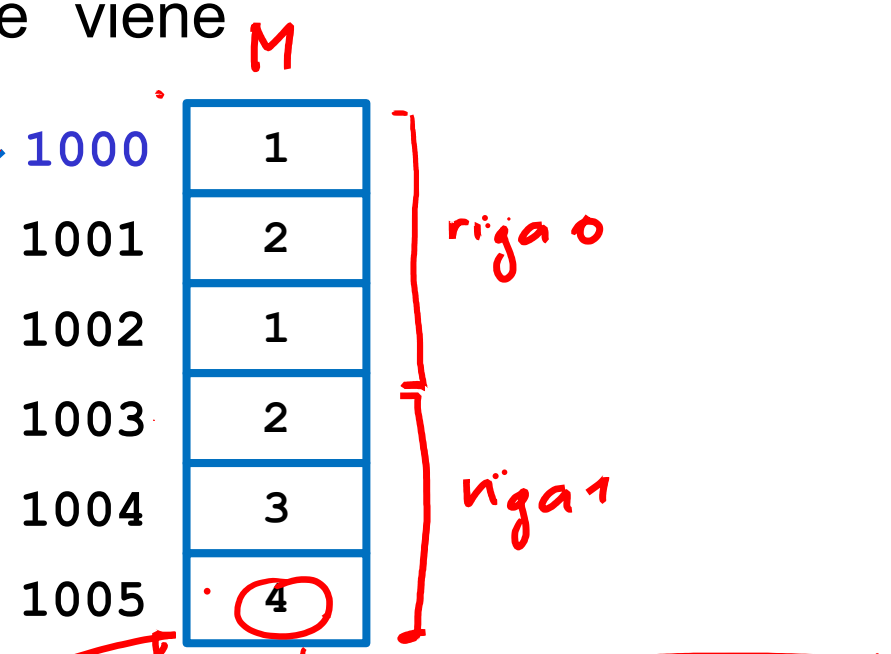
```
char funzione(int a[2][3])
{ return a[1][2];
}

main()
{ char x;
  char M[2][3]={{1,2,1},{2,3,4}};
  x = funzione(M);
}
```

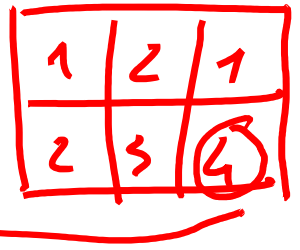
1000

1000

Per calcolare la cella giusta:

```
1000 +
1*3  +
 2   =
----
1005
```





Quindi è fondamentale sapere il numero di colonne della matrice (3), mentre non serve il numero delle righe (2)

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],...)
```

opz.

```
void g (int a[]){
...
}
```

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

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

DIMENSIONE LOGICA E FISICA

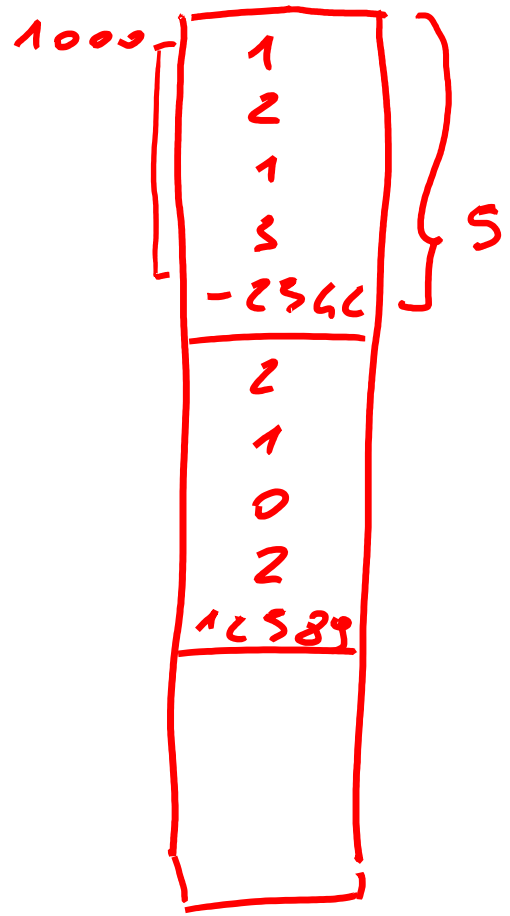
- Se non so la dimensione della matrice?
- sovradimensiono la matrice e poi ne uso solo una parte

Ncol = 4

Nrighe = 3

1	2	1	3	-2342
2	1	0	2	12389
4	2	3	1	-8238
-3243	-5343	-5234	2352	14234
13423	23421	24411	-1321	-2341

```
main()
{ int i,j, Nrighe, Ncol;
  int M[5][5];
  scanf("%d %d",&Nrighe,&Ncol);
  for(i=0;i<Nrighe;i++)
      for(j=0;j<Ncol; j++)
          scanf("%d",&M[i][j]);
}
```



ESERCIZIO

di interi

- Si legga da tastiera una matrice 3x3, tramite una procedura **`lettura`**
  n x n
- Si verifichi, tramite una funzione **`simm`**, se la matrice è simmetrica
  int simm (<matrice>)
- Si visualizzi se la matrice è simmetrica o no

1) leggere dim (es. 5)
2) leggere matrice (es. 5x5)

M

simm (M) → 1

1	2	1
2	4	5
1	5	2

N

1	2	1
2	4	5
1	6	2

simm (N) → 0