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];**

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]);
}
```

ARRAY E TYPEDEF

Si possono anche definire array di array:

Es. `typedef int riga[4];`

`typedef riga matrice[3];`

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

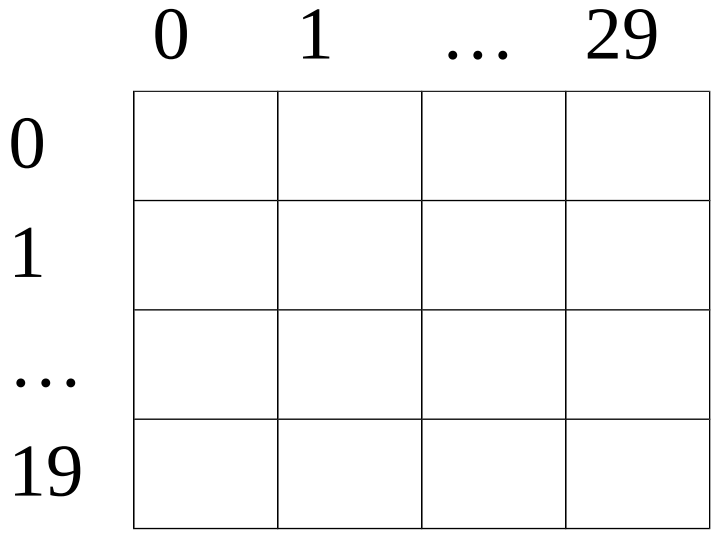
ARRAY MULTIDIMENSIONALI

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

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

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]

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

- parto dall'indirizzo della cella `a[0][0]`
- aggiungo `i` moltiplicato per la lunghezza di una riga
- sommo `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

ESERCIZIO

Si calcoli la matrice identità 3x3

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

PASSAGGIO DI PARAMETRI

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

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

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

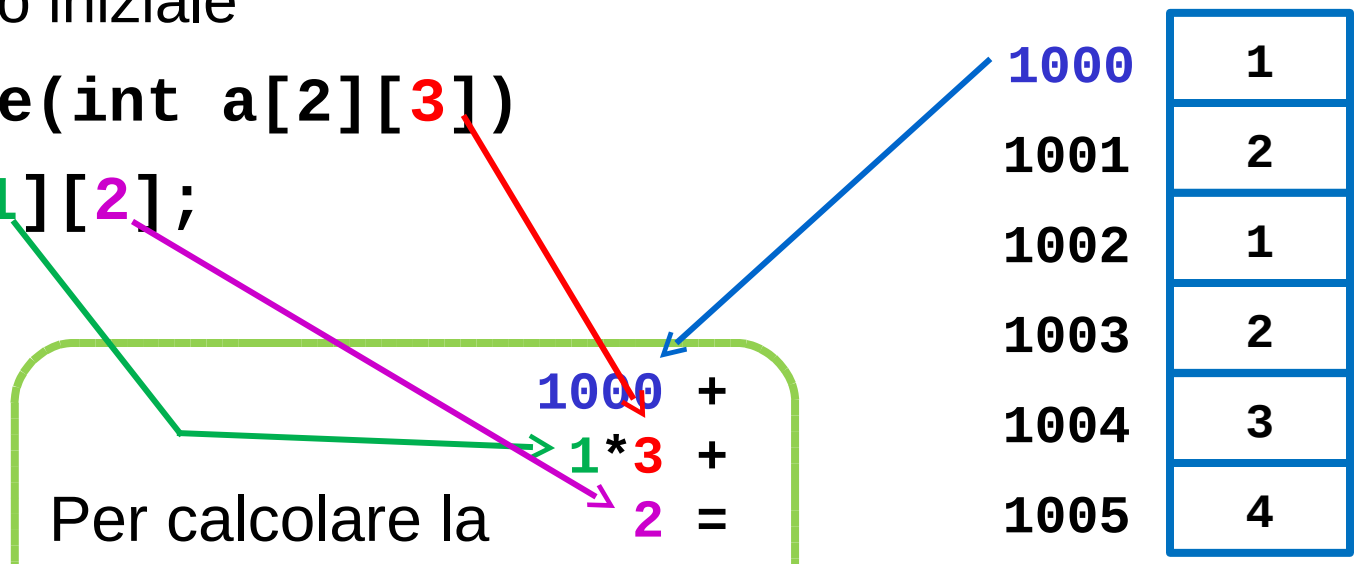
Nella definizione devo dichiarare il tipo

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

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




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

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],…)`

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***

ESERCIZIO

- Si legga da tastiera una matrice 3x3, tramite una procedura **`lettura`**
- Si verifichi, tramite una funzione **`simm`**, se la matrice è simmetrica
- Si visualizzi se la matrice è simmetrica o no

1	2	1
2	4	5
1	5	2

ESERCIZIO

- Si leggano da tastiera due matrici 2x2 A e B
- Si calcoli la matrice prodotto C = AxB
- Si visualizzi la matrice prodotto

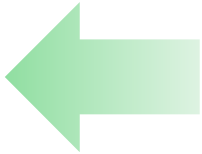
Esempio: PRODOTTO MATRICI QUADRATE

```
#define N 2
typedef int Matrici[N][N];

int main(void)
{ Matrici A,B,C;

  lettura(A);
  lettura(B);

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

  stampa(C);
}
```

Definiamo **Matrici** *come array NxN di* **int**

Esempio: PRODOTTO MATRICI QUADRATE

```
void stampa_riga(int riga[])
{ int j;
  for (j=0; j<N; j++)
  printf("%d\t",riga[j]);
  printf("\n");
}
void stampa(Matrici A)
{ int i;
  for (i=0; i<N; i++)
  stampa_riga(A[i]);
}
void lettura(Matrici A)
{ int i,j;
  for (i=0; i<N; i++)
  for (j=0; j<N; j++)
  scanf("%d",&A[i][j]);
}
```



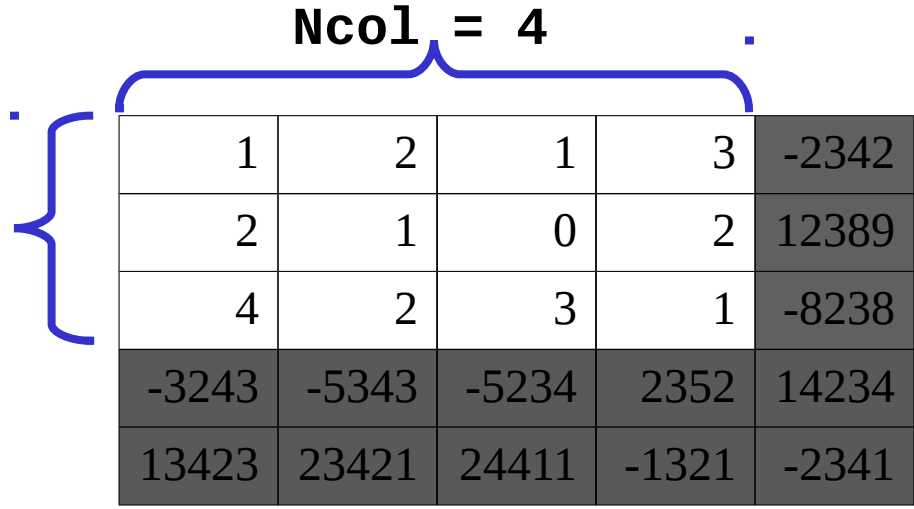
Passo una riga della matrice: ogni riga è un array di interi

Esempio: PRODOTTO MATRICI QUADRATE

```
void prodotto(Matrici X, Matrici Y, Matrici Z)
{
  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;
      }
}
```

DIMENSIONE LOGICA E FISICA

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




```
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

- Tramite una procedura, si leggano da tastiera
  - due interi: Nrighe e Ncolonne
  - una matrice Nrighe x Ncolonne
- Si scriva poi una procedura che calcola la matrice trasposta di quella inserita
- Infine, si stampi la trasposta