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];
```

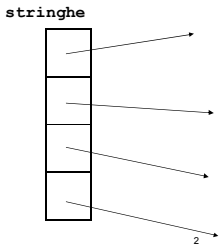
definisce un vettore di 4 puntatori a carattere (allocata memoria per 4 puntatori)

1

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***



2

INIZIALIZZAZIONE

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

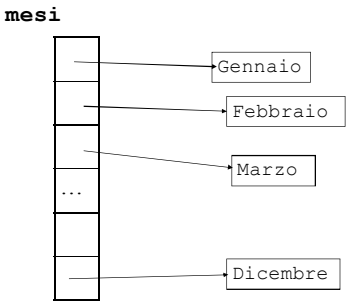
```
char * mesi[] =          {"Gennaio","Febbraio",
      "Marzo","Aprile","Maggio","Giugno",
      "Luglio","Agosto", "Settembre","Ottobre",
      "Novembre","Dicembre"};
```

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

3

INIZIALIZZAZIONE



4

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]);
}
```

5

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]);
}
```

6

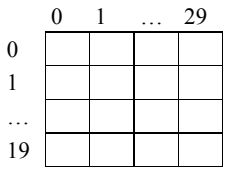
ARRAY MULTIDIMENSIONALI

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];
```

8

MATRICI

Le matrici vengono memorizzate per righe

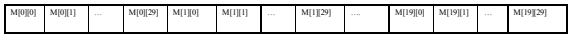
```
int a[3][4];
```

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

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



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)

10

MEMORIZZAZIONE

In generale, se

```
<tipo> mat[dim1][dim2]…[dimn]
```

```
mat[i1][i2]…[in-1][in]
```

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

11

ESERCIZIO

Si calcoli la matrice identità 3x3

12

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;
```

13

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

14

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

15

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***

16

PUNTATORI E VETTORI MULTIDIMENSIONALI

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

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

17

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]);
```

18

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

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);
}
```

1000	1
1001	2
1002	1
1003	2
1004	3
1005	4

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],…)`

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]);
}
```

A[0]	A[0][0]	A[0][1]
A[1]	A[1][0]	A[1][1]

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

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

- 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