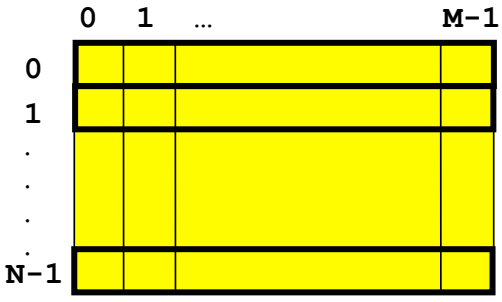
Matrici
(vettori multidimensionali)



matrice NxM

Matrici (vettori multidimensionali)

Gli elementi di un vettore possono essere a loro volta di tipo vettore: in questo caso si parla di *matrici*

Definizione di matrici:

```
<id-tipo> <id-variabile> [dim1] [dim2]..[dimN];
```

Significato:

- **`<id-variabile>`** è il nome di una variabile di tipo vettore di **`dim1`** componenti, ognuna delle quali è
  - un vettore di **`dim2`** componenti, ognuna delle quali è
    - un vettore di …., ognuna delle quali è
      - un vettore di **`dimN`** componenti di tipo **`<id-tipo>`**.

➔ Se le dimensioni sono N>1, il vettore si dice ***matrice N-dimensionale***

➔ (o 'matrice' per N=2; 'matrice quadrata' se dim1 = dim2)

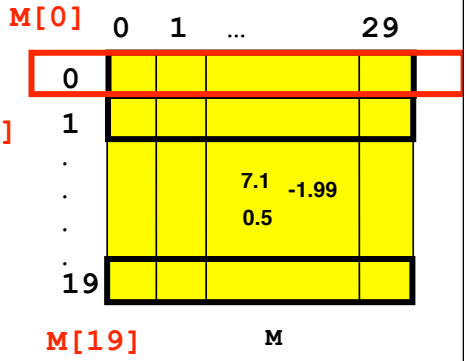
Vettori multidimensionali

Ad esempio: *matrice* di float

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

➔ M è un vettore di 20 elementi, ognuno dei quali è un vettore di 30 elementi, ognuno dei quali è un float.

➔ **`M[0]` rappresenta il primo vettore di 30 float (la prima *riga*)**

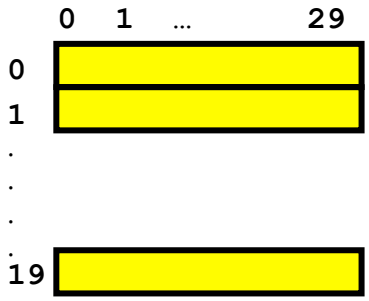


Accesso alle componenti:

il primo indice denota la ***riga***, il secondo la ***colonna***

```
M[0][0] = 7.1 ;
M[1][29] = 0.5 ;
M[19][29] = -1.99 ;
```


Matrici “per righe”



le colonne “non esistono”…

Dichiarazione di tipo per vettori multi-dimensionali

```
typedef  <id-tipo>  <id-tipo-vettore>[dim1][dim2].. [dimN];
```

Esempi:

```
typedef  float MatReali [20] [30];
MatReali Mat;
/*    Mat è un vettore di venti elementi, ognuno dei
  quali è un vettore di    trenta reali; quindi: Mat è
  una matrice di 20 X 30 di reali*/
```

In alternativa si puo` fare:

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


Inizializzazione di matrici

Anche nel caso di vettori multi-dimensionali l'inizializzazione si può effettuare in fase di definizione, tenendo conto che, in questo caso, gli elementi sono a loro volta vettori:

Esempio:

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

➔ La memorizzazione avviene "per righe":

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

Si può anche omettere il numero di righe:

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

Esempio

(lettura e stampa di matrici)

```
#include <stdio.h>
#define R 10
#define C 25

typedef float matrice[R][C];
main()
{      matrice M;
     int i, j;
     /* lettura: */
     for(i=0; i<R; i++)
          for(j=0; j<C; j++)
          {      printf("M[%d][%d]? ", i, j);
                 scanf("%f", &M[i][j]);        }
     /*stampa: */
     for(i=0; i<R; i++)
     {    for(j=0; j<C; j++)
                 printf("%f\t", M[i][j);
          printf("\n");    }
}
```


Esempio

(prodotto di due matrici)

Si realizzi un programma che esegua il prodotto (righe x colonne) di 2 matrici matrici a valori reali.

Definizione:

date due matrici rettangolari A[N][L] e B[L][M], il risultato di AxB è una matrice C[N][M] tale che:
∀ i ∈[0,N-1], ∀ j ∈[0,M -1]:

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

Rappresentazione dei dati:

```
/*costanti per le dimensioni delle matrici:*/
#define N 2
#define M 3
#define L 4

float A[N][L];
float B[L][M];
float C[N][M];
```

Soluzione

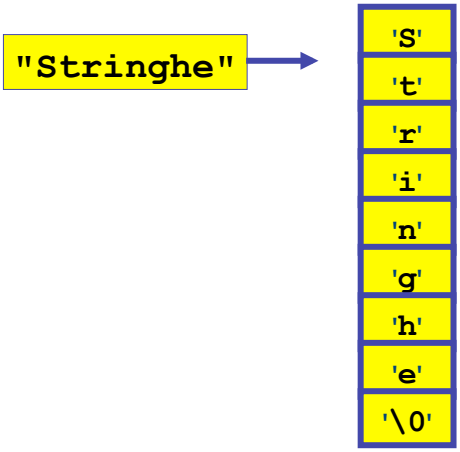
```
#include <stdio.h>
#define N 2
#define M 3
#define L 4
main()
{  float A[N][L];
   float B[L][M];
   float C[N][M];
   int i, j, k;
/* inizializzazione di A e B */
printf("Elementi di A?\n");
for (i=0; i<N; i++)
{  printf("\nRiga %d (%d elementi): ", i, L);
   for (j=0; j<L; j++)
     scanf("%f",&A[i][j]);
}
printf("Elementi di B?\n");
for (i=0; i<L; i++)
{  printf("\nRiga %d (%d elementi): ", i, M);
   for (j=0; j<M; j++)
     scanf("%f",&B[i][j]);
} /* continua.. */
```



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

/* stampa  del risultato: */
   for (i=0; i<N; i++)
   {      printf("\nC[%d]:\t", i);
          for (j=0; j<M; j++)
                 printf("%f\t",C[i][j]);
   }
}
```

Stringhe
(vettori di caratteri)



Vettori di caratteri: le stringhe

Una stringa è un vettore di caratteri, manipolato e gestito secondo la **convenzione**:

l'ultimo elemento significativo di ogni stringa è seguito dal carattere nullo '\0'
'\0' corrisponde al codice ASCII zero

È **responsabilità del programmatore** gestire tale struttura in modo consistente con il concetto di stringa (ad esempio, garantendo la presenza del terminatore '\0').

Ad esempio:

```
char A[10]={'b','o','l','o','g','n','a','\0'};
```

'b'	'o'	'l'	'o'	'g'	'n'	'a'	'\0'	?	?

oppure:

```
char A[10]="bologna"; /*          il terminatore '\0' è aggiunto automaticamente
   */
```

Lettura di stringhe (formato %s):

```
char B[25];

scanf("%s", &B); /* la sequenza di caratteri letta viene assegnata a B;
                    il terminatore '\0' è aggiunto automaticamente */
```

gets & puts

La scanf (con formato `%s`) prevede come separatore anche il blank (spazio bianco):

➔ è possibile soltanto leggere stringhe che non contengono bianchi;

Ad esempio, se l'input è:

Nel mezzo del cammin di nostra vita,
mi ritrovai per una selva oscura…
…

Leggendo con:

```
char s1[80], s2[80];
scanf("%s", s1); /* s1 vale "Nel" */
scanf("%s", s2); /* s2 vale "mezzo" */
```

➔ la `scanf` non è adatta a leggere intere linee (che possono contenere spazi bianchi, caratteri di tabulazione, etc.).

Per questo motivo, in C esistono funzioni specifiche per fare I/O di linee:

- `gets (legge fino a '\n')`
- `puts (scrive e aggiunge '\n')`

gets

è una funzione standard, che legge una intera riga da input, fino al primo carattere di fine linea (`'\n'`, newline) e l'assegna a una stringa.

```
gets(char str[]);
```

assegna alla stringa **`str`** i caratteri letti.
Il carattere `'\n'` viene sostituito (nella stringa di destinazione **`str`**) da `'\0'`.

Ad esempio, dato l'input:

Nel mezzo del cammin di nostra vita,
mi ritrovai per una selva oscura…

Leggendo con:

```
char S[80]; gets(S);
```

S vale:

"Nel mezzo del cammin di nostra vita,"

puts

è una funzione standard che scrive una stringa sull'output aggiungendo un carattere di fine linea (`'\n'`, newline):

```
puts(char str[]);
```

Ad esempio:

```
char S1[80]="Dante Alighieri";
char S2[80]="La Divina Commedia";
puts(S1);
puts(S2);
```

stampa sullo standard output:

```
Dante Alighieri
La Divina Commedia
```

➔ In generale: `puts(S);` è **equivalente** a `printf("%s\n", S);`

Esempio: lunghezza di una stringa

Programma che calcola la ***lunghezza*** di una stringa (cioè, il numero di caratteri significativi).

```
/* lunghezza di una stringa */
#include <stdio.h>
main()
{  char str[81]; /* str ha al max. 80 caratteri*/
   int i;
   printf("\nImmettere una stringa:");
   scanf("%s",&str); /* %s formato stringa */
   for (i=0; str[i]!='\0'; i++);
    printf("\nLunghezza: \t %d\n",i);
}
```

➔ La `scanf` legge i caratteri in ingresso fino al primo separatore (bianco, newline, ecc.) e li assegna al vettore `str` aggiungendo il terminatore di stringa.

NB: La lunghezza di una stringa si può anche calcolare utilizzando la funzione standard di libreria `strlen()` (previa inclusione di `string.h`)

Esempio: concatenamento di stringhe

Programma che concatena due stringhe s1 e s2 date: il risultato viene inserito in s1.

```
#include <stdio.h>
/* concatenamento di due stringhe */
main()
{
   char   s1[81], s2[81];
   int    j,i;
printf("\nPrima stringa: \t");
scanf("%s",&s1);
printf("\nSeconda stringa: \t");
scanf("%s",&s2);
for ( j=0 ; s1[j]!='\0' ; j++ );
for ( i=0 ; s2[i]!='\0' && i+j<79 ; i++ )
     s1[i+j]=s2[i];
s1[i+j]='\0';  /* fine stringa */
printf("\nStringa:\t%s\n",s1);
}
```

1 2 3

NB: il concatenamento di due stringhe si può anche ottenere utilizzando la funzione standard di libreria **`strcat()`** (previa inclusione di **`string.h`**)

3

string.h

Il C fornisce una libreria standard di funzioni per la gestione di stringhe; per poterla utilizzare è necessario includere il file header **`string.h`**:

```
#include <string.h>
```

Lunghezza di una stringa:

```
int strlen(char  str[ ]);
```

➔ restituisce la lunghezza (cioè, il numero di caratteri significativi) della stringa **`str`** specificata come argomento.

Ad esempio:

```
char S[10]= "bologna";
int k;
k=strlen(S);       /* k vale 7 */
```

string.h

Concatenamento di 2 stringhe:

```
strcat( char str1[], char str2[] );
```

➔ concatena le 2 stringhe date str1 e str2. Il risultato del concatenamento è in str1.

Ad esempio:

```
char S1[20] = "reggio";
char S2[20] = "emilia";
strcat( S1, S2 ); /* S1 = "reggioemilia" */
```

Copia di stringhe:

```
strcpy(char str1[], char str2[]);
```

➔ copia la stringa str2 in str1.

Ad esempio:

```
char S1[10] = "Riccardo";
char S2[10];
strcpy( S2, S1 ); /* S2 = "Riccardo" */
```


string.h

Confronto di 2 stringhe:

```
int strcmp(char str1[ ], char str2[ ]);
```

➔ esegue il confronto tra le due stringhe date str1 e str2. Restituisce:

- 0 se le due stringhe sono identiche;
- un valore negativo (ad esempio, -1), se str1 precede str2 (in ordine lessicografico);
- un valore positivo (ad esempio, +1), se str2 precede str1 (in ordine lessicografico).

Ad esempio:

```
char S1[10] = "bologna";
char S2[10] = "napoli";
int k;
k = strcmp(S2,S1); /* k > 0 */
k = strcmp(S1,S2); /* k < 0 */
k = strcmp(S2,S2); /* k = 0 */
```

string.h

strlen(char s[])

→ |s1|

strcpy(char s1[], char s2[])

→ *(assegnamento)* s1=s2

strcat(char s1[], char s2[])

→ *(somma)* s1 = s1+s2

strcmp(char s1[], char s2[])

→(s1 == s2) ? 0 : ((s1 < s2) ? -1 : +1);

Il linguaggio C
Il record

Adrian	Ailincai	2900	21.0
Marco	Albergamo	31000	27.0
Leonardo	Amico	8000	21.0
Massimiliano	Arpino	1500	19.2
Valerio	Zerillo	45000	34.5

Il Record

Esempio:

Si vuole rappresentare adeguatamente l'astrazione "contribuente", caratterizzata dai seguenti attributi:

- **Nome,**
- **Cognome,**
- **Reddito,**
- **Aliquota**

Con gli strumenti visti finora, per ogni contribuente è necessario introdurre 4 variabili:

```
char   Nome[20], Cognome[20];
int    Reddito;
float  Aliquota;
```

È una soluzione scomoda e non "astratta": le quattro variabili sono indipendenti tra di loro.

➔ È necessario un costrutto che consenta l'aggregazione dei 4 attributi nell'astrazione "contribuente": il record.

Tipi strutturati: il Record

Un record è un insieme finito di elementi, in generale non omogeneo:

- il numero degli elementi è rigidamente fissato a priori.
- gli elementi possono essere di tipo diverso.
- il tipo di ciascun elemento componente (campo) è prefissato.

Ad esempio:

NOME	COGNOME	REDDITO	ALIQUOTA

Formalmente:

Il record è un tipo strutturato il cui dominio si ottiene mediante prodotto cartesiano:

dati n insiemi, A_{c1}, A_{c2}, ..., A_{cn}, il prodotto cartesiano tra essi:

$$A_{c1} \times A_{c2} \times \ldots \times A_{cn}$$

consente di definire un tipo di dato strutturato (il record) i cui elementi sono n-ple ordinate:

$$(a_{c1}, a_{c2}, \ldots, a_{cn})$$

dove $a_{ci} \in A_{ci}$.

Ad esempio:

Il **numero complesso** può essere definito attraverso il prodotto cartesiano R × R.

Il Record in C: `struct`

Collezioni con un numero finito di campi (anche disomogenei) sono realizzabili in C mediante il costruttore di tipo strutturato **`struct`**.

Definizione di variabile di tipo struct:

```
struct {
  <lista def. campi>
  }  <id-variabile>;
```

```
struct {
       char  Nome[20],
             Cognome[20];
       int   Reddito;
       float Aliquota;
       } c1;
```

dove:

- **`<lista def. campi>`** è l'insieme delle definizioni dei campi componenti, costruita usando le stesse regole sintattiche della definizione di variabili:

```
<tipo1> <campo1>;
<tipo2> <campo2>;
...
<tipoN> <campoN>;
```

```
char  Nome[20],
      Cognome[20];
int   Reddito;
float Aliquota;
```

- **`<id-variabile>`** è l'identificatore della variabile di tipo record così definita.

Il tipo struct

Accesso ai campi:

Per accedere (e manipolare) i singoli campi di un record si usa la notazione ***postfissa*** :

<id-variabile>.<componente>

indica il valore del campo **<componente>** della variabile **<id-variabile>** .

➔ I singoli campi possono essere manipolati con gli operatori previsti per il tipo ad essi associato.

```
struct {
       char  Nome[20],
             Cognome[20];
       int   Reddito;
       float Aliquota;
       } c1;
```

`c1.Nome`

```
strcpy(c1.Nome,"Valentina");
strcpy(c1.Cognome,"Florio");
c1.Reddito = 7000+18000;
c1.Aliquota = 23.0;
```

Il tipo struct: assegnamento

```
struct {
       char   Nome[20],
              Cognome[20];
       int    Reddito;
       float  Aliquota;
       } c1, c2;
```

```
strcpy(c1.Nome,"Valentina");
strcpy(c1.Cognome,"Florio");
c1.Reddito = 7000+18000;
c1.Aliquota = 23.0;

c2=c1;
```

Operatori:

L'unico operatore previsto per dati di tipo struct è l'operatore di ***assegnamento*** (=):

➔è possibile l'assegnamento diretto tra record di tipo equivalente.

Inizializzazione di record:

È possibile inizializzare i record in fase di definizione.

Ad esempio:

```
struct {   char   Nome[20],
                  Cognome[20];
           int    Reddito;
           float         Aliquota;
 } c1 ={"Valentina", "Florio", 7000+18000, 23.0 };
```

typedef struct

Il costruttore struct può essere utilizzato per dichiarare **tipi non primitivi** basati sul record:

Dichiarazione di tipo strutturato record:

```
typedef struct {
<lista dichiarazioni campi>
} <id-tipo>;
```

dove:

- `<lista definizioni campi>` è l'insieme delle definizioni dei campi componenti;
- `<id-tipo>` è l'identificatore del nuovo tipo.

Ad esempio:

```
typedef struct {  char  Nome[20],
                       Cognome[20];
                  int   Reddito;
                  float Aliquota;
                } contribuente;
contribuente c1,c2,c3;
strcpy(c1.Nome, "Valentina");
```

< lista dichiarazione campi >

< id-tipo >

typedef struct

Nota: non è consentito definire due tipi diversi con lo stesso nome.
Però è possibile che un campo del record abbia lo stesso nome di una variabile. Come `Reddito`, nell'esempio:

```
typedef struct {  char Nome[20];
                  char Cognome[20];
                       int Reddito;
                  float Aliquota;
                } contribuente;
contribuente c1, c2, c3;

unsigned int Reddito = 7000;

c1.Reddito = Reddito + 18000;
```

variabile.campo (variabile strutturata) **variabile** (intero senza segno)

Il tipo struct in sintesi

Riassumendo, la sintassi da adottare è:

```
[typedef] struct {
        <tipo-1> <nome_campo-1>;
        <tipo-2> <nome_campo-2>;
                ...
        <tipo-N> <nome-campo-N>;
           }  <nome>;
```

Vincoli:

- **<nome-campo-i>** è un identificatore stabilito che individua il campo i-esimo;
- **<tipo-i>** è un qualsiasi tipo, semplice o strutturato.
- **<nome>** è l'identificatore della struttura (o del tipo, se si usa **typedef**)

Uso:

- la struttura è una collezione di un numero fissato di elementi di vario tipo (**<tipo-campo-i>**);
- il singolo campo **<nome-campo-i>** di un record **R** è individuato mediante la notazione: **R.<nome-campo-i>;**
- se due strutture di dati di tipo **struct** hanno lo stesso tipo, allora è possibile l'assegnamento diretto.