Operatore condizionale

Operatore condizionale:

<condizione> ? <parteVera> : <parteFalsa>

restituisce il valore di <parteVera> o di <parte Falsa>, in base al seguente criterio:

- la <parteVera> viene valutata solo se la <condizione> è verificata (valore diverso da 0)
- la <parteFalsa> viene valutata solo se la <condizione> non è verificata (valore uguale a zero)

Esempio:

```
ris=a>b? a:b;
```

equivale a:

```
if(a>b)
    ris=a;
  else
    ris=b;
```

Operatori sui bit

Operatori sui bit: consentono di modificare la rappresentazione delle variabili (**`int`** o compatibili):

<< shift a sinistra es: k<<4 shift a sinistra di 4 bit (equivale a k*16)

>> shift a destra es: k>>4 shift a destra di 4 bit (equivale a k/16)

`&` and bit a bit

`|` or *inclusivo* bit a bit

`^` or *esclusivo* bit a bit

`~` complemento a 1

Istruzioni per il trasferimento del controllo: `break` e `continue`

Istruzione `break`:

L'istruzione **`break`** provoca l'uscita immediata dal ciclo (o da un'istruzione **`switch`**) in cui è racchiusa.

Esempio:

```
for (F=1, i=1; ; i++)/* manca il controllo */
   if (i>N) break;
   else F=F*i;
```

Istruzione `continue`:

L'istruzione **`continue`** provoca l'inizio della successiva iterazione del ciclo in cui è racchiusa (non si applica a **`switch`**).

Esempio:

```
for (sum=0, i =1; i <= N; i++)
   if (i%2) continue;
   else  sum+=i;
```

Tipi di dato strutturati

I dati strutturati (o strutture di dati) sono ottenuti mediante composizione di altri dati (di tipo semplice, oppure strutturato).

Tipi strutturati in C:

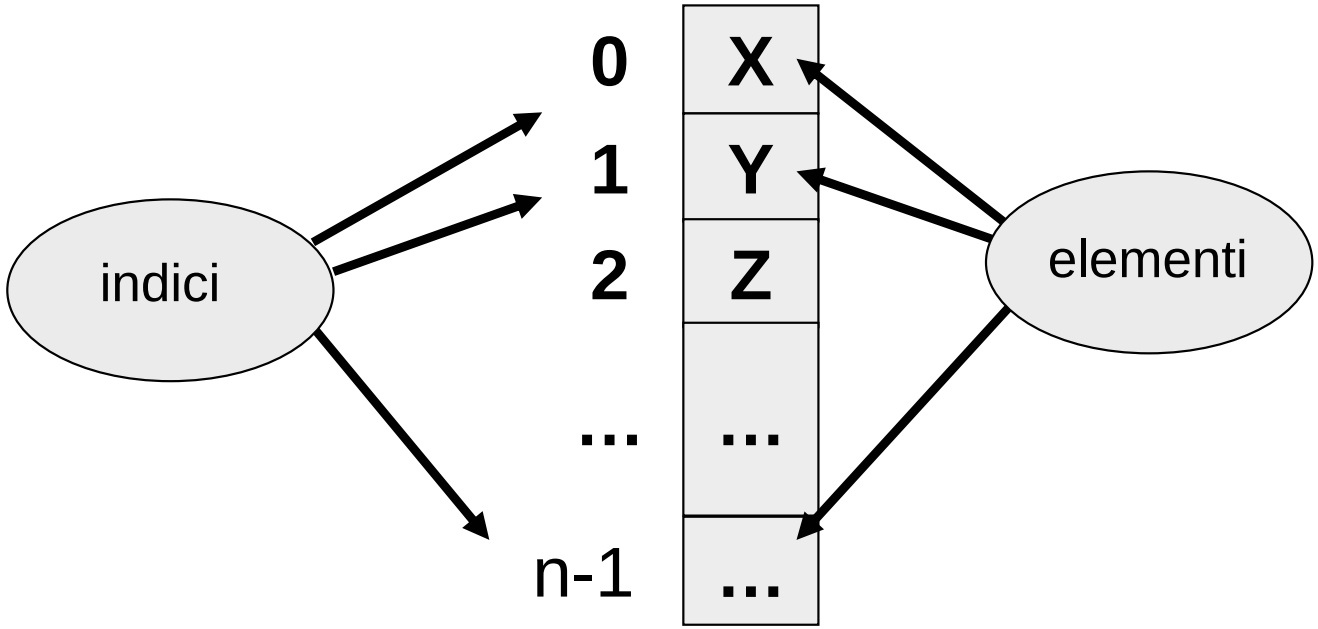
- **vettori** (o ***array***)
- **record** (***struct***)
- [record varianti (*union*)]

Vettori (o *array*)

Un **vettore** (o ***array***) è un insieme ordinato di elementi tutti dello stesso tipo.

Caratteristiche del vettore:

- omogeneità
- ordinamento, ottenuto mediante dei valori interi (***indici***) che consentono di accedere ad ogni elemento della struttura.



Definizione di vettori in C

Nel linguaggio C per definire vettori, si usa il ***costruttore di tipo*** **[ ].**

Sintassi:

<id-tipo> <id-variabile> [<dimensione>];

dove:

- **<id-tipo>** è l'identificatore di tipo degli elementi componenti,
- **<dimensione>** è una costante intera che rappresenta il numero degli elementi componenti,
- **<id-variabile>** è l'identificatore della variabile strutturata (il vettore) così definita

Esempio:

```
int    V[10];  /* vettore di dieci elementi interi */
```

NB: La dimensione (numero di elementi del vettore) deve essere una **costante intera**, nota al momento della dichiarazione.

```
int    N;
char   V[N];  /* def. sbagliata!!! */
```

Vettori in C

```
int V[25]; /*def. di un vettore di 25 elementi interi*/
```

Indici:

- ad ogni elemento è associato univocamente un ***indice*.**
- è possibile riferire una singola componente specificando l'indice **`i`** corrispondente, utilizzando la notazione **`V[i]`.**
- L'indice deve essere di tipo enumerabile (ad esempio: **`int, char`**).
- se **N** è la **dimensione** del vettore (numero degli elementi), il dominio degli indici è l'intervallo

[0,N-1]

Esempio:

```
float  A[3], i; /* A è un vettore di          */
A[0]=22.05; /*ass. di un valore al primo elemento */
A[2]=17.9; /*ass. di un valore al        elemento */
A[1]=0; /*ass. di un valore al          elemento */
```

A

Vettori in C

Operatori:

In C **non esistono operatori specifici** per i vettori.

Per operare sui vettori è necessario operare singolarmente sugli elementi componenti (coerentemente con il tipo ad essi associato).

Pertanto:

➔ **non è possibile l'assegnamento diretto** tra vettori:

```
int V[10], W[10];
…
V=W;    /* è scorretto! */
```

➔ **non è possibile leggere (o scrivere)** un intero vettore (fanno eccezione, come vedremo, le stringhe); occorre leggere/scrivere le sue componenti;

Esempio: lettura di un vettore:

```
int V[100];
int  i;

for(i=0; i<100; i++)
   {            printf("valore %d-simo? ", i);
                scanf("%d", &V[i]);
   }
```

Gestione degli elementi di un vettore

Le singole componenti di un vettore possono essere elaborate con gli operatori del tipo ad esse associato.

Esempio: agli elementi di un vettore di interi è possibile applicare tutti gli operatori definiti per gli interi:

```
int A[10], n=7 i=2;
A[i] =n%i;              /* A[ ]=   */
A[9]=A[i++]+7;          /* A[9]=       i=   */
scanf("%d", &A[i]);     /*ass. ad A[ ]                    /
```

`typedef … … [ ]`

In C è possibile utilizzare **`typedef … … [ ]`** per ***dichiarare*** tipi di dato non primitivi rappresentati da vettori.

Sintassi della dichiarazione:

```
typedef <tipo-componente> <tipo-vettore> [<dim>];
```

dove:

- **`<tipo-componente>`** è l'identificatore di **tipo di ogni singola componente**
- **`<tipo-vettore>`** è l'identificatore che si attribuisce al **nuovo tipo**
- **`<dim>`** è il **numero di elementi** che costituiscono il vettore (deve essere una costante).

Esempio:

```
typedef int Vettore[30];
Vettore  V1,V2;  /* V1 e V2 sono variabili di tipo
                    Vettore; ognuno rappresenta un vettore
                    di 30 elementi interi. */
```

➔ **V1 e V2 possono essere utilizzati come vettori di interi.**

Vettori in sintesi

Riassumendo:

- **Definizione di variabili di tipo vettore:**

```
<tipo-componente> <nome-variabile>[<dim>];
```

- **Dichiarazione di tipi** non primitivi rappresentati da vettori:

```
typedef  <tipo-componente>   <nome-tipo>   [<dim>];
```

Caratteristiche:

- **`<dim>`** è una costante enumerabile.
- **`<tipo-componente>`** è un qualsiasi tipo, semplice o strutturato.
- il vettore è una sequenza di dimensione fissata **`<dim>`** di componenti dello stesso tipo **`<tipo-componente>`.**
- la singola componente i-esima di un vettore V è individuata dall'indice i-esimo, secondo la notazione **`V[i]`.**
- L'intervallo di variazione degli indici è **`[ 0 , … , dim-1 ]`**
- sui singoli elementi è possibile operare secondo le modalità previste dal tipo **`<tipo-componente>`.**

Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

2 possiblità. 1) <u>Mediante un ciclo</u>:

Per attribuire un valore iniziale agli elementi di un vettore, si può attuare con una sequenza di assegnamenti alle N componenti del vettore.

Esempio:

```
#define N 30
typedef int vettore [N];
vettore v;
int i;
...
for( i=0; i<N; i++ )
  v[i]=0;
```

`#define`: La **`#define`** è una direttiva del *preprocessore* C che serve ad associare un identificatore (nell'esempio N) ad una costante (nell'esempio, 30). Rende il programma più facilmente modificabile.

Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

2) In alternativa al ciclo, è possibile l'inizializzazione in fase di definizione:

Esempio:

```
int v[10] = {1,2,3,4,5,6,7,8,9,10};

/*  v[0] = 1; v[1]=2;...v[9]=10; */
```

Addirittura è possibile fare:

```
int v[]={1,2,3,4,5,6,7,8,9,10};
```

➔ La dimensione è determinata sulla base dell'inizializzazione.

Esempio
(somma di due vettori)

Problema:

Si realizzi un programma che, dati da standard input gli elementi di due vettori A e B, entrambi di 10 interi, calcoli e stampi gli elementi del vettore C (ancora di 10 interi), ottenuto come somma di A e B.

- Dichiariamo il nuovo tipo di dato **`vettint`**:

```
;
```

- Definiamo i due vettori dati A e B, e il vettore C risultante dalla somma :

```

```

```
#include <stdio.h>
typedef int vettint[10];

main()                                  risultato
{  vettint A, B, C;
   int i;
   for(i=0; i<10; i++) /* lettura del vettore A */
   {   printf("valore di A[%d] ? ", i);
       scanf("%d", &A[i]);
   }
   for(i=0; i<10; i++) /* lettura del vettore B */
   {   printf("valore di B[%d] ? ", i);
       scanf("%d", &B[i]);
   }

   for(i=0; i<10; i++)                  /* calcolo del risultato */
       C[i]=A[i]+B[i];

   for(i=0; i<10; i++) /* stampa del risultato C */
       printf("C[%d]=%d\n",i, C[i]);
}
```

Esempio

(calcolo del numero medio di iscritti per classe)

Si consideri una scuola media costituita da 4 sezioni (da 'A' a 'D'), ognuna costituita da tre classi (“livello” 1, 2, 3).

Dati gli iscritti ad ogni classe di ogni sezione, si vuole calcolare il numero medio di studenti per classe relativo ad ogni sezione.

Definiamo un nuovo tipo che rappresenta il **`livello`**:

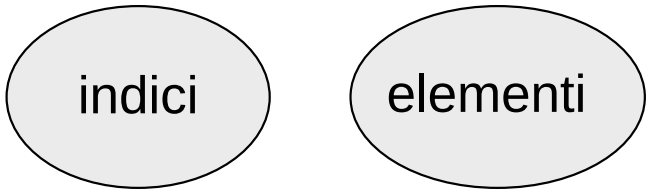
```
/* un elemento per ogni sezione */

typedef
livello prime, seconde, terze;
```

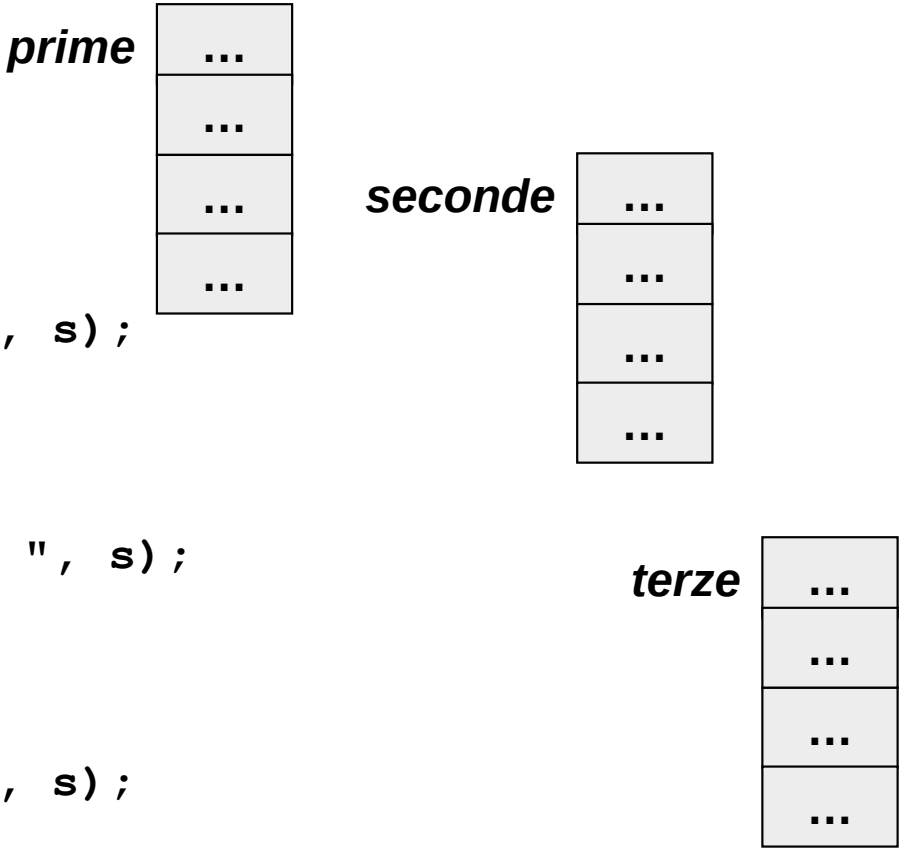
`s='A', s-'A' = 0`	**n0**
`s='B', s-'A' = 1`	**n1**
`s='C', s-'A' = 2`	**n2**
`s='D', s-'A' = 3`	**n3**

NB: L’indice di un vettore deve essere di tipo ***enumerabile*** ➔ può essere un **`char`**:

```
typedef char sezione;
sezione s;  // s = 'A', 'B', …
/*indice per accedere ai vettori delle classi */
```



```
#include <stdio.h>
typedef int livello[4]; /*un elemento per ogni sezione */
typedef char sezione;
main()
{  livello prime, seconde, terze;
   sezione s;
   float  media=0;
   for(s='A'; s<='D'; s++)
   {    printf("Iscritti alla prima %c: ", s);
        scanf(                          );
   }
   for(s='A'; s<='D'; s++)
   {    printf("Iscritti alla seconda %c: ", s);
        scanf(                          );
   }
   for(s='A'; s<='D'; s++)
   {    printf("Iscritti alla terza %c: ", s);
        scanf(                          );
   }
   for(s='A'; s<='D'; s++, media=0)
   {    media =                                                    ;
        printf("\nLa media degli iscritti per classe
                        nella sezione %c è: %f\n", s, media );
   }
}
```

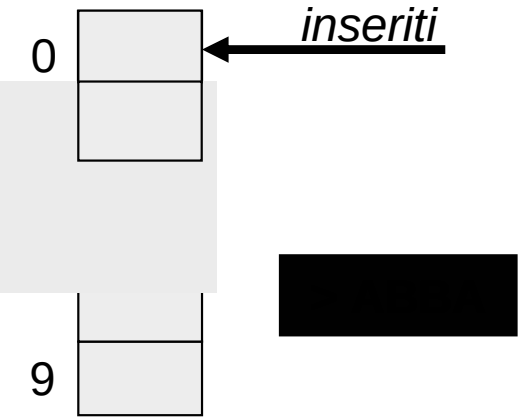


Esempio
(stampa caratteri senza ripetizioni)

Si leggano da input alcuni caratteri alfabetici maiuscoli (hp: al massimo, 10) e si riscrivano in uscita evitando di ripetere caratteri già stampati.

Soluzione:

while <ci sono caratteri da leggere>
{
};
while <ci sono elementi della struttura dati>
<stampa elemento>;



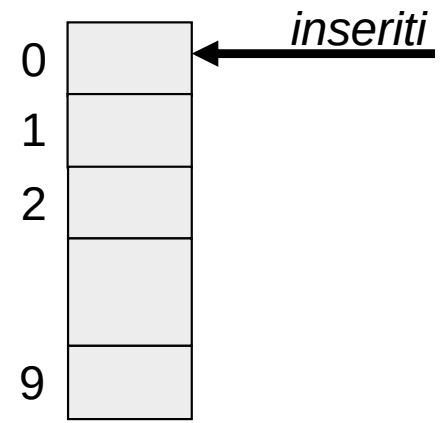
➔ Occorre una variabile dove memorizzare (senza ripetizioni) gli elementi letti in ingresso: il ***vettore A***

➔ il vettore A verrà riempito con al più 10 valori: definiamo la variabile ***inseriti***, che rappresenterà la ***"****dimensione* ***logica"*** di A (cioè, il numero di elementi significativi):

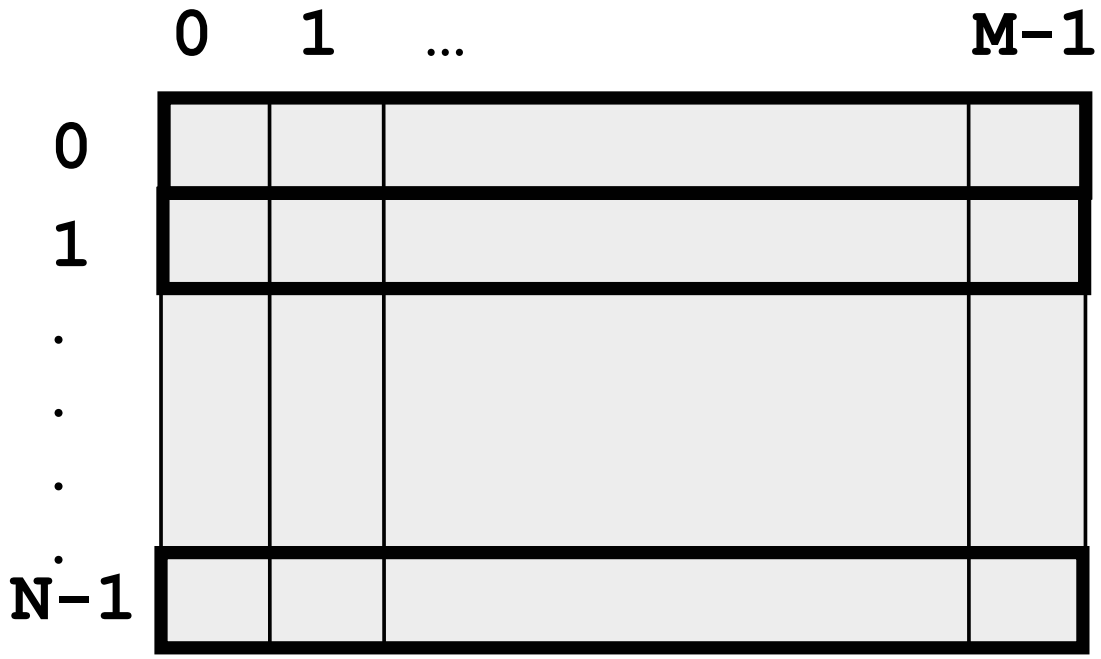
➔ *la dimensione* ***fisica*** *del vettore è 10, mentre la dimensione* ***logica*** *è data dal numero effettivo di caratteri diversi tra loro*

Soluzione

```
#include <stdio.h>
main()
{  char A[10], c;
   int i, j, inseriti=0, trovato;
   printf( "\n Dammi 10 caratteri: " );
   for ( i=0; ( i<          ); i++ )
   {   scanf("%c", &c);
       /* verifica unicità: */
       trovato=0;
       for( j=0 ;                          ; j++ )
              if (c==A[j])
                     trovato=1;
       /* se c non è ancora presente in A, lo inserisco: */
       if (          )
       {
       }
   }
   printf( "Inseriti %d caratteri \n", inseriti );
   for (                                )
       printf( "%c\n", A[i] );
}
```



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>`**.

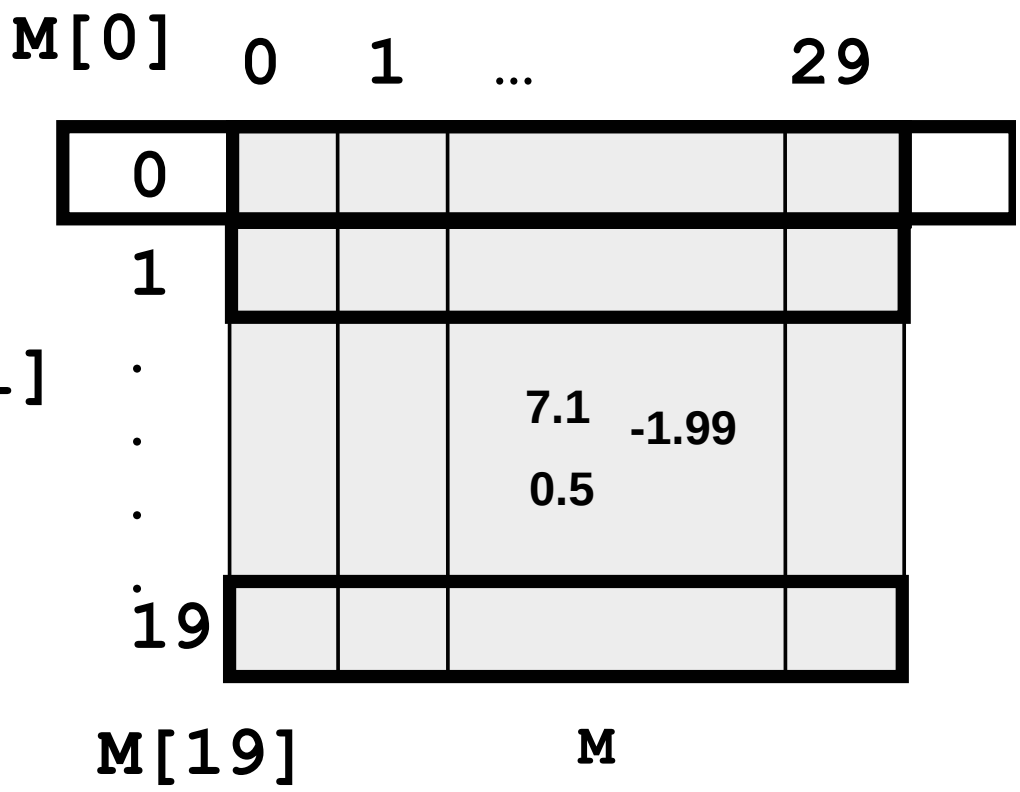
➔ Se le dimensioni sono N>1, il vettore si dice ***matrice N-dimensionale***

➔ (‘matrice’ o 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.

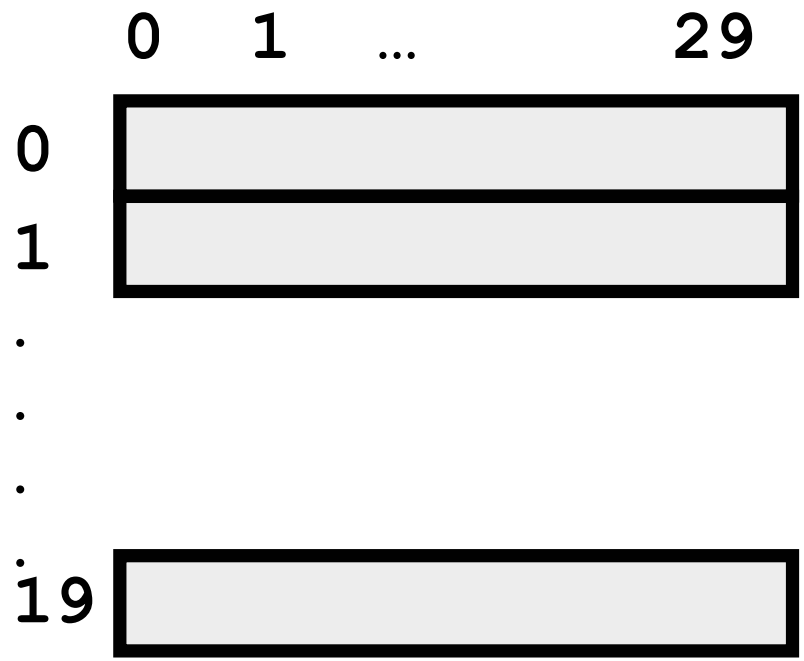
Accesso alle componenti:

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 ;
```

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

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},
                    {0,0,1,0},
                    {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	0	1	0
3	0	0	0	1

Si può anche omettere il numero di righe:

```
int matrix[][4]={{1,0,0,0},{0,1,0,0},{0,0,1,0},{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",          );         }
        /*stampa: */
        for(i=0; i<R; i++)
        {       for(j=0; j<C; j++)
                        printf("%f\t",         );
                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 e` una matrice C[N][M] tale che:

$\forall$ **i** $\in$**[0,N-1],** $\forall$ **j** $\in$**[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(                );
}
```

```
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(                );
} /* continua.. */
```

```
/* ...prodotto matriciale:  */
for (i=0; i<N; i++)
   for (j=0; j<M; j++)
   {
   }
```

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

Esempio
ordinamento di un vettore

Dati n valori interi forniti in ordine casuale, stampare in uscita l'elenco dei valori dati in ordine crescente.

➔ E` necessario mantenere in memoria tutti i valori dati per poter effettuare i confronti necessari: utilizziamo i vettori.

Soluzione:

Esistono vari procedimenti risolutivi per questo problema (algoritmi di ordinamento, o ***sorting***):

- **naïve-sort**
- bubble sort
- quick sort
- merge sort
- ...

Risolveremo il problema utilizzando il Metodo dei Massimi successivi (*naïve-sort*).

Ordinamento di un vettore mediante *naïve sort*

Descrizione dell' algoritmo:

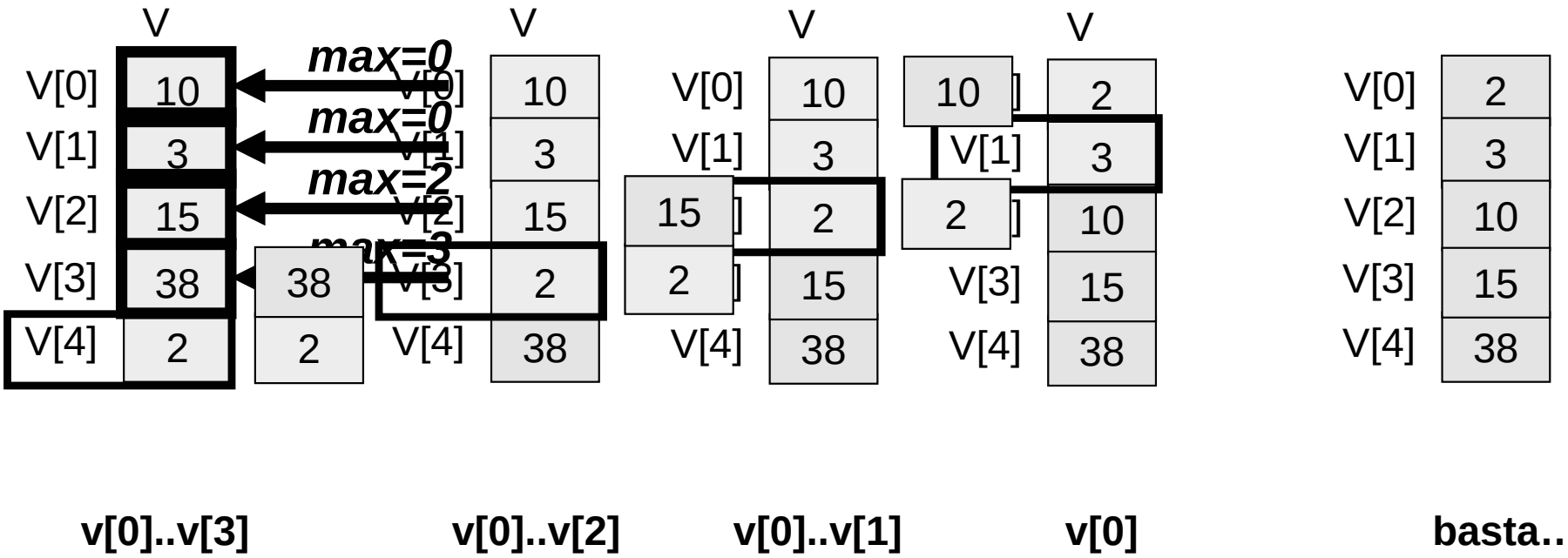
Dato un vettore: **`int V[dim];`**

1. eleggi un elemento come massimo temporaneo (**`V[max]`**)
2. confronta il valore di **`V[max]`** con tutti gli altri elementi del vettore (**`V[i]`**):

   se **`V[i]>V[max] , max=i`**
3. quando hai finito i confronti, scambia **`V[max]`** con **`V[dim-1];`** il massimo ottenuto dalla scansione va in ultima posizione.
4. riduci il vettore di un elemento (**`dim=dim-1`**) e, se **`dim>1`**, torna a 1.

naïve sort

con vettore di dimensione `dim = 5`



```
#include <stdio.h>
#define dim 10
main()
{  int V[dim], i,j, max, tmp, quanti;
   /* lettura dei dati */
   for ( i=0; i<dim; i++ )
   {   printf("valore n. %d: ",i);
       scanf("%d", &V[i]);
   }
   /*ordinamento */
   for(  ciclo di n= dim-1 iterazioni )
   {
```

determinazione del
valore massimo presente
nella porzione di vettore

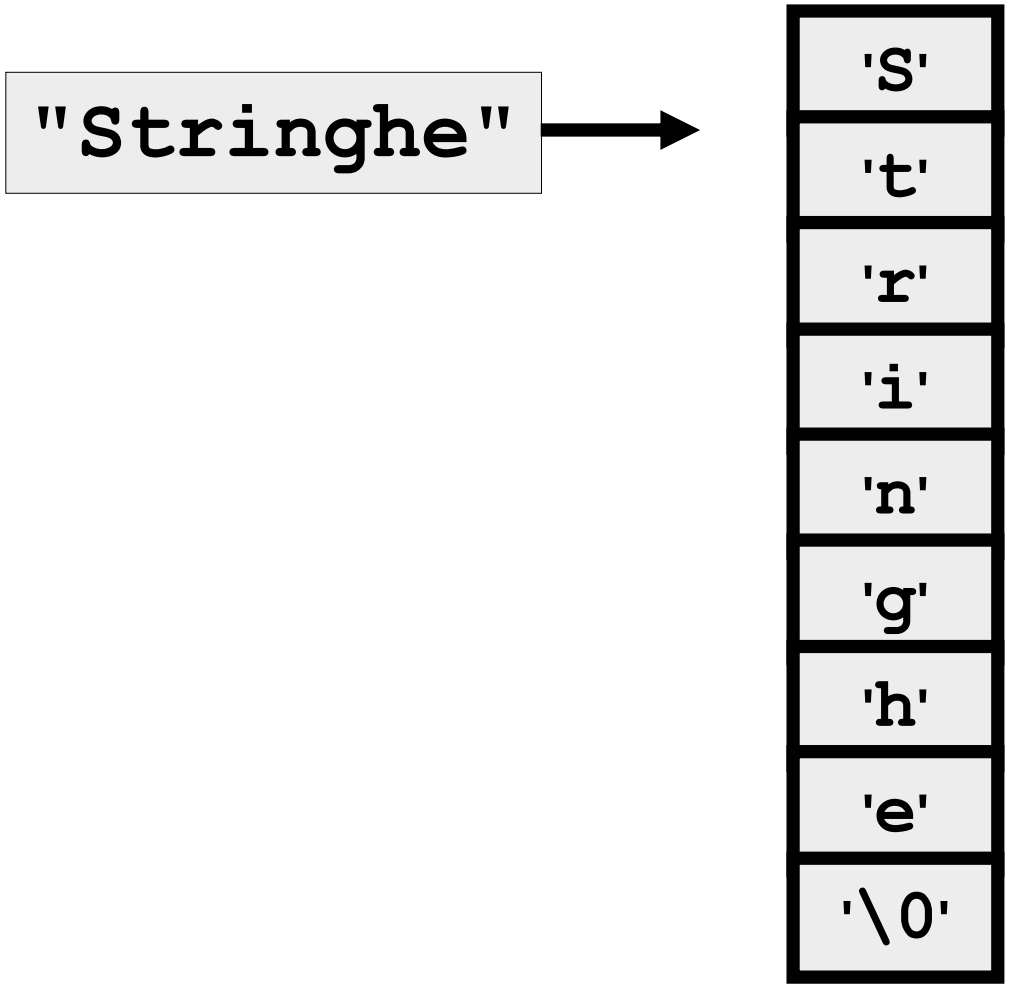
se ho trovato un valore maggiore
di quello presente nell'ultima cella
effettuo uno scambio

```
   }
/* stampa del vettore ordinato per esercizio */
```

Soluzione

```
/* ..stampa il vettore ordinato */
   for (i=0; i<dim; i++)
       printf("\n%d", V[i]);
}
```

Stringhe
(vettori di caratteri)



Vettori di caratteri: le stringhe

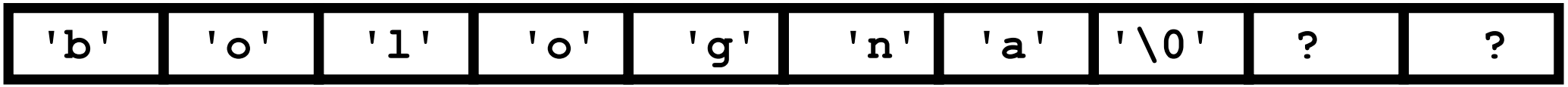
Una stringa è un vettore di caratteri, manipolato e gestito secondo la **convenzione**:

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'};
```



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 */
```

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);
}
```

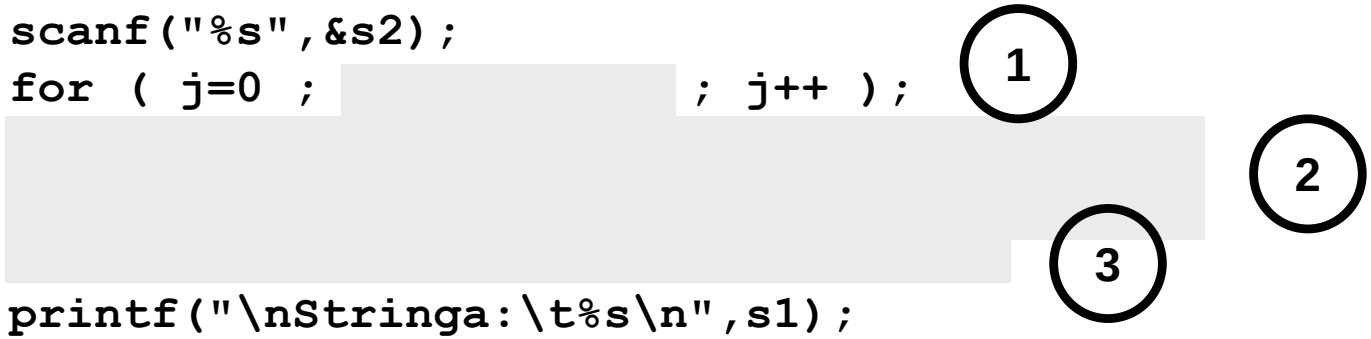
➔ La **`scanf`** legge i caratteri in ingresso fino al primo separatore (bianco, newline, ecc.) e li assegna al vettore **`str`** aggiungendo il terminatore di stringa.

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 ;                ; j++ );
printf("\nStringa:\t%s\n",s1);
}
```



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

Libreria standard sulle stringhe

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*/
```

Libreria standard sulle stringhe

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" */
```

Libreria standard sulle stringhe

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(S1,S2); /* k < 0 */
k=strcmp(S1,S1); /* k = 0 */
k=strcmp(S2,S1); /* 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);*

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 in 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

e` una funzione standard, che legge una intera riga da input, fino al primo carattere di fine linea (`'\n'`, newline) e l'assegna ad 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 in 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);`