

array:
vettori, matrici e stringhe

Vettori

- Un vettore (array) è una sequenza di variabili tutte dello stesso tipo che occupano locazioni di memoria contigue.
- Dichiarazione di un **vettore** di oggetti del tipo **Tipo**:
Tipo **identificatore**[**dimensione**];
- Uso di un elemento del **vettore**:
identificatore[**elemento**]
- Dove **dimensione** e **elemento** sono degli oggetti di tipo **unsigned int**

NB

- Diversamente da altri linguaggi in C e C++ **il primo elemento di un vettore ha indice 0** e l'ultimo, se n è la dimensione del vettore, ha indice $n-1$.
- Come in tutti i linguaggi la dichiarazione del vettore ed il suo dimensionamento comportano **l'allocazione della memoria necessaria per la dimensione dichiarata**: se si prova ad usare un elemento con indice superiore a $n-1$ si ha uno sfondamento della memoria, errore che comporta problemi in punti imprevedibili del programma e che non può essere messo in evidenza dal compilatore.

Esempi di dichiarazioni

```
int V[25]; /*dichiarazione di un vettore V  
          con 25 elementi integer  
          V[0],V[1]...V[24]. */
```

```
double vector[3]; /* dichiarazione di un vettore  
                  vector con 3 elementi double  
                  vector[0],vector[1],vector[2]. */
```

```
char c[50]; /* dichiarazione di un vettore c  
            con 50 elementi character  
            c[0],...,c[49] */
```

attenzione!

- la dimensione del vettore deve essere nota al momento della sua dichiarazione.
- Il seguente codice (che generalmente non produce errore in compilazione!) e' sbagliato:

```
int n, data[n];  
printf("inserire la lunghezza del vettore data ");  
scanf("%d",&n);
```

- e' invece corretto specificare la dimensione massima del vettore e usarne solo una parte:

```
int nmax=100, data[n];  
printf("inserire la lunghezza del vettore (<100) ");  
scanf("%d",&n);
```

- meglio ancora se si utilizza il preprocessore

```
#define NMAX 100  
int data[NMAX];
```

Esempi con inizializzazione

le seguenti dichiarazioni riservano lo spazio in memoria per il vettore e lo inizializzano

```
int V[25] = {3, 5, 6, 1} ; /* i restanti elementi sono posti a 0 */
```

```
double vector[3] = {-1.2, 4.7, 5.9} ;
```

```
char word[50] = {'s', 'p', 'o', 't'}; /* i restanti elementi sono posti a 0 */
```

```
int M[ ] = {1, 4, 15, 2}; /* il vettore viene dimensionato con 4 elementi */
```

```
int k[100]={0}; /* tutti gli elementi vengono inizializzati a zero */
```

Esempio di inizializzazione e uso

```
double A[6] = {4.1, 5.3, 10.6, 100.9, 23, -12};  
int j;  
double sum = 0;  
  
for(j = 0; j < 6; j++) sum = sum + A[j];  
  
printf("la somma vale %f\n", sum);
```

Vettori multidimensionali (matrici)

- Dichiarazione di una matrice con
 - **rDim** righe
 - **cDim** colonne

Tipo identificatore[**rDim**][**cDim**];

- Esempio:

```
double tabella[3] [4];
```

Dichiarazione ed inizializzazione

Per l'inizializzazione si elencano gli elementi della prima riga, poi della seconda etc...

```
int M[3][4] = { {1,2,3,4},  
                {5,6,7,8},  
                {9,10,11,12}};
```

oppure

```
int M[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

Esempio: prodotto scalare

```
#include <stdio.h>
main(){
    float A[3],B[3],pScal=0;
    int j;
    printf("inserisci gli elementi dei vettori A e B\n");
    for(j=0; j<3; j++) {
        printf("A[%d]=",j);
        scanf("%f",&A[j]);
    }
    for(j=0; j<3; j++) {
        printf("B[%d]=",j);
        scanf("%f",&B[j]);
    }
    for(j=0; j<3; j++) {
        pScal += A[j]*B[j];
    }
    printf("A.B=%f\n",pScal);
}
```

Esempio di esecuzione del programma:

```
inserisci gli elementi dei vettori A e B
A[0]=1
A[1]=0
A[2]=0
B[0]=2.5
B[1]=1
B[2]=1
A.B=2.500000
```

Esempio: modulo di un vettore

```
#include <stdio.h>
#include <math.h>
main(){
    double A[3],pScal=0;
    int j;
    printf("inserisci gli elementi del vettore A\n");
    for(j=0; j<3; j++) {
        printf("A[%d]=" ,j);
        scanf("%lf",&A[j]);
    }
    for(j=0; j<3; j++) {        pScal += A[j]*A[j];        }
    printf(" |A|=%lf\n",sqrt(pScal));
}
```

Esempio: prodotto vettoriale

```
#include <stdio.h>
main(){
    int j;
    double A[3]= {1.,0.,0. };
    double B[3]= {0.,1.,0. };
    double C[3];
    int permutazioni[3][2]={    {1,2},    {2,0},    {0,1}};

    for(j=0; j<3; j++) {
        int k=permutazioni[j][0];
        int m=permutazioni[j][1];
        C[j]=A[k]*B[m]-A[m]*B[k];
        printf("C[%d]=%lf\n",j,C[j]);
    }
}
```

Esempio: prodotto di matrici

```
#include <stdio.h>
/* | 1 0 |
   | 2 1 | * | 0 1 2 |
   | 0 2 |   | 1 2 1 | */
main( ) {
    int M1[3][2]= { {1,0}, {2,1}, {0,2}};
    int M2[2][3]= { {0,1,2}, {1,2,1}};
    int M3[3][3];
    int j,k,l;
    for(j=0; j<3; j++) { /* righe di M1 */
        for(k=0; k<3; k++) { /* colonne di M2 */
            M3[j][k]=0;
            for(l=0; l<2; l++ ) { /* colonne di M1 */
                M3[j][k]+= M1[j][l]*M2[l][k];
            }
        }
    }
    for(j=0; j<3; j++) {
        for(k=0; k<3; k++) {
            printf("%d ",M3[j][k]);
        }
        printf("\n");
    }
}
```

risultato stampato

```
0 1 2
1 4 5
2 4 2
```

Stringhe

- Una stringa è rappresentata da un vettore di caratteri la cui dimensione deve essere pari almeno al numero di caratteri+1 (il carattere di terminazione della stringa '\0')
- La seguente dichiarazione corrisponde ad un vettore di 54 stringhe ciascuna avente al massimo 50 caratteri

```
char studenti[54][50];
```

- esempi di inizializzazione:

```
char Name[ ] = "Michael Bird";
```

Equivalente a

```
char Name[ ] = {'M','i','c','h','a','e','l',' ','B','i','r','d','\0'};
```

O a

```
char Name[13] = {'M','i','c','h','a','e','l',' ','B','i','r','d','\0'};
```

Ovvero

```
Name[0] = 'M' , Name[1] = 'i' ... Name[11] = 'd' , Name[12] = '\0'
```

I/O per le stringhe di caratteri

- il descrittore di formato per i caratteri e' `%c`, per le stringhe e' `%s`. Se pero' proviamo a leggere una stringa contenente degli spazi con `scanf` scopriamo che questa funzione legge solo i caratteri che precedono il primo spazio. Inoltre non viene effettuato nessun controllo sulla lunghezza della stringa inserita: e' quindi possibile provocare uno sfondamento di memoria.

```
main(){
    char l[10];
    scanf("%s",l); /* attenzione, l in questo caso NON deve
                   essere preceduto da & */
    printf("%s",l);
}
```

- per ovviare a questi problemi e' necessario leggere i caratteri uno per uno mediante `getchar()` (che richiede l'inclusione dell'header `stdlib.h`) come indicato nell'esempio 5.8 del testo di Barone et al. e in quello di seguito riportato.

uso di getchar()

```
#include <stdio.h>
#include <stdlib.h>
#define NMAX 10

main() {
    char c, line[NMAX];
    int i = 0;
    printf("Immettere la stringa in input: ");
    while ((c = getchar()) != '\n') {
        if (i < NMAX-1) {
            line[i++] = c;
        }
        else {
            printf("Raggiunto limite massimo della stringa\n");
            exit(1);
        }
    }
    printf("%s",line);
}
```