**35. Array multidimensionali**
In C++ è possibile dichiarare array i cui elementi siano a loro volta degli array, generando degli array multidimensionali (matrici, se sono array bidimensionali).

Sintassi
*tipo id [dim1] [dim2] [dim3] … [dimn];*
*tipo id[dim1][dim2]...[dimN]={lista_valori};*
*tipo id[][dim2]...[dimN]={lista_valori};*

Un array multidimensionale *dim1 ·…·dimn* può essere pensato come un array di *dim1* array multidimensionali *dim2 ·…·dimn*.

I più usati sono gli array bidimensionali, ovvero le matrici (la prima dimensione indica le righe, la seconda le colonne).

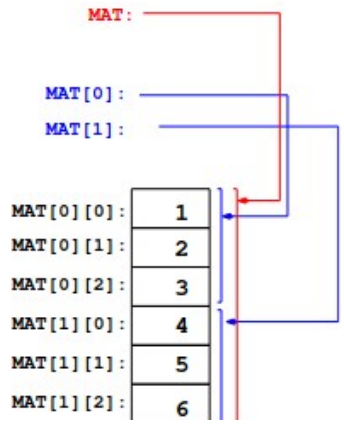
Esempio
*int MAT[2][3];*
*int MAT[2][3] = {{1, 2, 3}, {4, 5, 6}};*
*int MAT[][3] = {{1, 2, 3}, {4, 5, 6}};*

La prima dimensione (solo la prima!) può essere omessa e ricavata poi dalla lista valori.
**! Tutte** le dimensioni successive alla prima vanno dichiarate **!**
La lista valori è di questo tipo:

{ {, , , , ,}, {, , , , , }, {, , , , ,}, {, , , , ,}, {, , , , , } };
Tante sottoparentesi graffe quante sono le righe (=*dim1*) e tanti elementi in ogni sottoparentesi quante sono le colonne (=*dim2*).

Come per gli array le dimensioni devono essere valutate in fase di compilazione e, come per essi, gli elementi mancanti in inizializzazione vengono tutti inizializzati a zero di default.

**Struttura di un array bidimensionale (statico).** `int MAT[2][3] = {{1,2,3},{4,5,6}};`




**35.1 Assegnazione e stampa di una matrice.**

Assegnazione di valori alla matrice

```
for(int m=0;m<righe;m++){
  for(int n=0;n<colonne;n++){
    matrice[m][m] = rand()%20;
  }
}
```

Stampa della matrice

```
for(int m=0; m<righe; m++){
  for(int n=0; n<colonne; n++){
    cout << matrice[m][n] << " ";
  }
  cout << endl << " ";
}
```

**Servono due *for* annidati per trattare tutti gli elementi di una matrice!**

**35.2 Passaggio di array bidimensionali a funzione.**
Quando passiamo una matrice come parametro formale ad una funzione, dobbiamo passare la matrice specificando la *dim2* e altre due variabili, che sono rispettivamente *d1* la dimensione virtuale delle righe e *d2* la dimensione virtuale delle colonne, che sono le variabili con cui effettivamente lavoriamo con la matrice.

È imperativo però passare la matrice in questo modo: ***matrice[][DIM];*** dove *DIM* è la dimensione assegnata in fase di inizializzazione della matrice e **definita globalmente!** Altrimenti nella dichiarazione non viene trovata. Anche le matrici vengono passate per riferimento.

Definizione: *void funz(int matrix[][DIM], int d1, int d2);*
Dichiarazione: *void funz(int [][DIM], int, int);*
Chiamata: *funz(matrix,d1,d2);*

**35.3 Dimensione virtuale**
Come per gli array è possibile determinare una dimensione virtuale delle matrici. Bisogna definire matrici molto grandi e poi chiedere all'utente le dimensioni (ovvero le righe e le colonne della matrice). Successivamente in tutte le operazioni useremo le dimensioni virtuali per effetuare operazioni.

Quando passeremo una matrice ad una funzione useremo come parametro formale *matrix[][DIM]* dove *DIM* è la dimensione originaria della matrice e passeremo anche d1, d2 che sono rispettivamente le righe e le colonne decise dall'utente della nostra matrice.

**35.4 Prodotto matriciale**

```
void product  (const float a[][Dim], const float b[][Dim], float c[][Dim], int d1, int d2, int d3)
{
   int i, j, k;

   for (i=0; i < d1 ; i++)
      for (j=0; j < d3 ; j++) {
           c[i][j]=0;
        for (k=0; k < d2; k++)
           c[i][j] += a[i][k]*b[k][j];
      }
}
```

*d1* rappresenta le righe della prima matrice
*d2* rappresenta le colonne della seconda matrice
*d3* rappresenta le colonne della prima e le righe della seconda

Altri programmi utili per le matrici sono: trovare il valore minimo e massimo degli elementi di una matrice (scorrere la matrice confrontando ogni elemento con *min* e *max*, che a priori vengono assegnati a *m[0][0]* per poi essere cambiati), sommare due matrici (dello stesso tipo, si ottiene una matrice dove i componenti sono la somma dei rispettivi componenti) e moltiplicare ogni elemento di una matrice per un dato valore.

Versione ricorsiva del prodotto matriciale

```
void prod(int a[][DIM], int b[][DIM], int ris[][DIM], int d1, int d3, int d2){
  prod(a,b,ris,d1,d3,d2,0,0,0);
}
```

```
void prod(int a[][DIM], int b[][DIM], int ris[][DIM], int d1, int d3, int d2, int i, int j, int k){

  if(i<d1){
    if(j<d2){

      if(k<d3){
        ris[i][j] += a[i][k]*b[k][j];
            prod(a,b,ris,d1,d3,d2,i,j,++k);
      } else {
            prod(a,b,ris,d1,d3,d2,i,++j,0);
      }

    } else {
      prod(a,b,ris,d1,d3,d2,++i,0,0);
    }
  }
}
```

Versione ricorsiva della stampa di una matrice
Prima della chiamata, fare un *cout << endl;*

```
void stampa(int matrix[][DIM], int row, int column){
  stampa(matrix,row,column,0,0);
}

void stampa(int matrix[][DIM], int row, int column, int m, int n){
  if(m<row){
    if(n<column){
      cout << matrix[m][n] << " ";
      stampa(matrix,row,column,m,++n);
    } else {
      cout << endl;
      stampa(matrix,row,column,++m,0);
    }
  }
}
```

Versione ricorsiva assegnazione

```
void assegna(int matrix[][DIM], int row, int column){
  assegna(matrix,row,column,0,0);
}

void assegna(int matrix[][DIM], int row, int column, int m, int n){
  if(m<row){
    if(n<column){
      matrix[m][n]=rand()%10;
      assegna(matrix,row,column,m,++n);
    } else {
      assegna(matrix,row,column,++m,0);
    }
  }
```

**36. Array e puntatori**

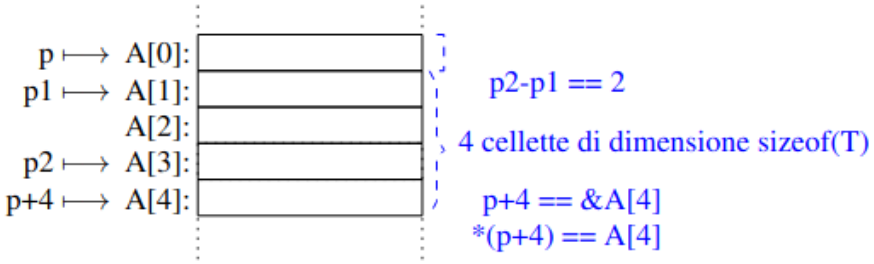
Se un puntatore *p* punta al primo elemento di un array *A*, l'espressione *p+i* è l'indirizzo dell'i-esimo elemento dell'array *A*.

*int A[DIM]={};*
*int* p=&A;*

*p+i == &(A[i])*
**(p+i) == A[i]*

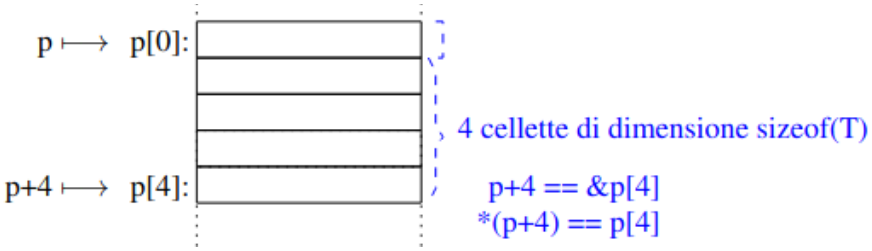
*p* contiene l'indirizzo di memoria della prima cella dell'array e siccome le celle in memoria sono consecutive, sommando un intero *i* a un vettore si passa alla cella che viene *i* celle dopo quella che punta il puntatore iniziale e di conseguenza all'elemento dell'array con indice *i*.

Se due puntatori dello stesso tipo *p1*, *p2* puntano ad elementi di uno stesso array, *p2-p1* denota il numero di elementi compresi tra *p1* e *p2* (*p2-p1* è negativo se *p2* precede *p1*).




Inoltre il nome di un array è <u>implicitamente equivalente</u> a una costante puntatore al primo elemento dell'array stesso.

Per questo se *p* è un array noi possiamo accedere all'elemento all'indice *i* sia facendo così *p[i]* che **(p+i)*. Anche le espressioni *&p[i]* e *p+i* sono equivalenti.



**36.1 Passaggio di array alle funzioni tramite puntatori**
Nelle chiamate di funzioni viene passato solo l'indirizzo alla prima locazione dell'array.
Un parametro formale array può essere specificato usando indifferentemente la notazione degli array o dei puntatori.

Per gli array semplici
*int f(int arr[dim]);*
*int f(int arr[]);*
*int f(const int *arr);*

Per gli array multidimensionali
*int f(int arr[dim1][dim2]...[dimN]);*
*int f(int arr[][dim2]...[dimN]);*
*int f(const int *arr[dim2]...[dimN]);*

Si può anche scrivere *int f(int []);* nella dichiarazione e *int f(int *);* nella definizione e viceversa.
Inoltre si può anche passare alla funzione che nella definizione presenta l'array, un puntatore all'array nella chiamata ad essa.

**36.1.2 Restituzione di un array**
Una funzione può definire un array allocato staticamente se e solo se è **allocato staticamente esternamente alla funzione** (come parametro formale o array globale). Non si può restituire un array che è stato dichiarato all'interno della funzione.

La funzione che ritorna un array, ritorna una variabile di tipo puntatore.

```
int array[dim]={};     //array globale

… int main (){
int v;
int w[dim]={};          //array del main
v=funz();
w=funz2();
v=funz3(v);    //non si può fare
}

int* funz(){
return a;
}

int* funz2(){
return w;
}
```