# Array Multidimensionali

Il C permette la creazione di array di qualunque tipo. In particolare è possibile usare array di array ecc.

una dichiarazione come:
int b[2][7]
definisce un'array bidimensionale di dimensione 2x7
int c[5][3][2]
definisce un'array tridimensionale di dimensione 2x3x5.

**Array Bidimensionali**
Quando consideriamo la memorizzazione di un array bidimensionale
è conveniente pensare agli elementi come organizzati come un rettangolo (Si noti che gli array multidimensionali non sono idonei nella loro forma normale alla memorizzazione di matrici.
int [2][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]

# Array Multidimensionali

Espressioni equivalenti a a[i][j] per i

*(a[i]+j)
(*(a+i))[j]
*((*(a+i))+j)
*(&a[0][0] +4*i+j)

Nota: le parentesi risultano necessarie perchè le parentesi quadre hanno priorità più alta rispetto all'operatore di indirizzamento *.

**Mappa di Memorizzazione**

int a[dim1][dim2]
La corrispondenza tra i puntatori e gli indici dell'array è chiamata
**mappa di memorizzazione**

a[i][j] = *(&a[0][0]+(dim2-1)*i+j)

# Array Multidimensionali

**Passaggio dei parametri**

Quando un array multidimensionale viene passato come parametro tutte le dimensioni eccetto la prima devono essere specificate nella dichiarazione. Ciò rende inadeguata l'uso di array multidimensionali per implementare matrici. Siamo sempre forzati a lavorare su matrici con tutte le dimensioni fissate.

```
int sum (int  a[][5])
```

la dichiarazione è totalmente analoga a :

```
int a[3][5]   int (*a)[5]
```

la prima dimensione viene ignorata dal compilatore.

Nel secondo le parentesi sono necessarie perché le parentesi quadre hanno priorità più alta rispetto a *.

Questa regola comunque non vale in generale: per esempio nell'intestazione di una funzione

char x[][] non è equivalente char **x;

# Array Multidimensionali

**Inizializzazione**

```
int a[2][3]={1,2,3,4,5,6}
inta[2][3]{{1,2,3},{4,5,6}}
```

Come per il caso unidimensionale se il numero di elementi inizliazzati è minore del numero di elementi del vettore il resto vengono inzializzati a 0.

Esempio: come eliminare troppi 0

```
int a[2][2][3] ={
        { {1,1,0},{2,0,0} }
        {{3,0,0},{4,4,0}}
        }
```

è equivalente a

```
int a[2][2][3] ={
        { {1,1},{2} }
        {{3},{4,4}}
        }
```

# Array Multidimensionali

**typedef e Matrici**

```
#define N 100
typedef double scalar;
typedef scalar vector[N];
typedef scalar matrix[N][N];
```

matrix A,B;
vector C,D;

somma di matrici
prodotto di matrici
prodotto scalare

# Array di puntatori

**Commentiamo un programma per ordinare le parole contenute in un file**

Creiamo un file "input" che contiene le seguenti parole:

"Domani parto per i caraibi"

Chiamando sort_words < input  otterremo il seguente output

caraibi

domani

i

parto

per

# Array di puntatori

**Commentiamo un programma per ordinare le parole contenute in un file**

```
#incllude <stdio.h>
#include <string.h>
#include<stdlib.h>

#define MAXWORD  100        /* lughezza massima di una parola */
#define N        1000       */ numero massimo parole */

void sort_word(char *w[], int n);
void swap (char **p, char **q);
void write_word(char *w[], int n);

#include "sort.h"
```

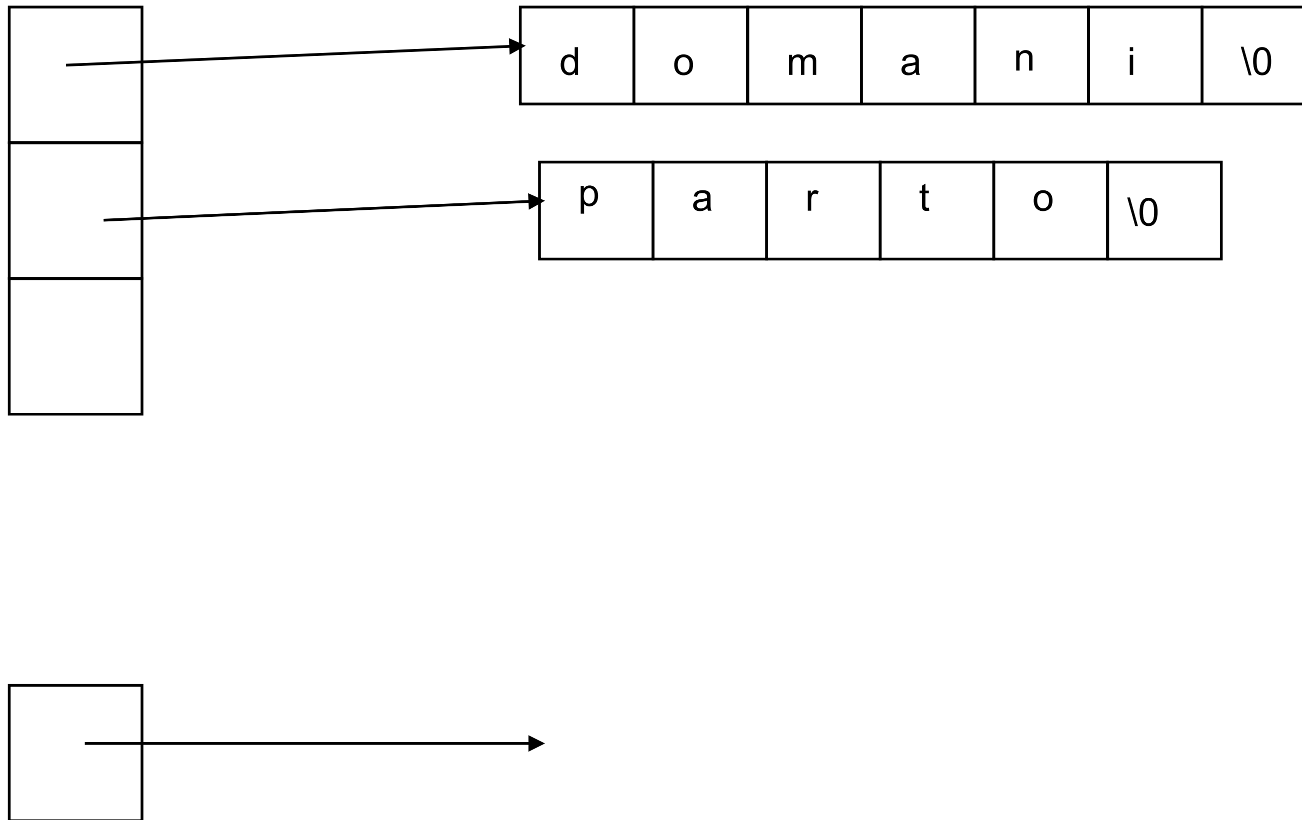
# Array di puntatori

```
#include "sort.h"

int main(void){
        char word[MAXWORD]   /* spazio di lavoro */
        char *w[N];          /* array di puntatori a caratteri */
        int n;               /* numero parola da ordinare */
        int i;

        for (i=0; scanf("%s",word)==1; ++i){
            if (i>=N) printf ("ERRORE: TROPPE PAROLE");
            if (strlen(word)>= MAXWORD) printf("ERRORE: PAROLA TROPPO LUNGA");
            w[i] = calloc(strlen(word)+1,sizeof(char));
            if (w[i]== NULL) printf("ERRORE di CALLOC");
            strcpy(w[i],word);
        }
        n=i;
        sort_word(w,n);
        write_words(w,n);
        return 0;
}
```

# Array di puntatori



# Array di puntatori

```
int sort_word(char *w, int n){
        int i,j;
        for (i=0; i<n; ++i){
            for(j=i+1;j<n; ++j){
                if (strcmp(w[i],w[j])>0)
                    swap(&w[i],&w[j]);
            }
        }
}
int strcmp(char *s1 , char *s2)
```

restituisce un intero minore uguale o maggiore di 0 in relazione al fatto che s1 sia lessicograficamente minore uguale o maggiore di s2.

```
swap(char **p, char **q){
        char *tmp;
        tmp =*p;
        *p = *q;
        *q = tmp
}
```

# Parametri del main

int main(int argc , char  *argv[])

argc contine il numero dei parametri sulla riga di comando
argv è un array di puntatori a char che puo essere visto come un array di stringhe ognuna delle quali è una parola sulla riga di comando.

Es:

./a.out  my echo is for apple

argc = 6
argv contiene le 6 parole della linea di comando.

# Funzioni come parametri

**Ma che cosa è un parametro funzione ???**
Il compilatore C interpreta un parametro di tipo funzione come un puntatore ad una funzione. Di fatto il parametro contiene l'indirizzo di inizio della zona di memoria dove è memorizzata la funzione.
La seguente intestazione di funzione è totalmente analoga alla precedente

```
double somma_quadrati ( double (*f)(double), int m, int n)
```

Si noti che

in **double (*f)(double)** le parentesi (*f) sono necessarie in quanto la priorità di () è piu alta di quella di *.

Se avessimo scritto:

**double *g(double);** avremmo dichiarato *g* come una funzione che riceve un double e restituisce un puntatore a double

# Funzioni come parametri

Supponiamo di dover scrivere un programma che calcoli $\sum_{k=m\ldots n} f(k)^2$ per diverse funzione f(k). per esempio f(k) = sin k o f(k) = 1/k.

Normalmente dovremmo scrivere due programmi: uno che calcola la somma per f(k) e l'latro per f(k) = sink.

Il *C* invece permette di semplificare questa situazione perchè consente di avere una funzione come parametro di una funzione:

```
double somma_quadrati( double f(double), int m, int n){

      int k,sum=0;
      for(k=m;k<=n;k++)
         sum += f(k)*f(k);
      return sum;
}
```

# Funzioni come parametri

Nel corpo della funzione somma_quadrati il puntatore ad f può essere trattato alla stregua di una funzione, oppure deferenziato esplicitamente.

Nella funzione avremmo potuto scrivere equivalentemente

```
sum += (*f)(k)*(*f)(k)
```

In questa espressione

f è un puntatore ad una funzione
*f è il nome della funzione
(*f)(k) è la chiamata alla funzione

# Funzioni come parametri

```
#include <stdio.h>
#include <math.h>

double  frac(double);
double somma_quadrati( double (*)(double), int , int);

int main(void){
printf("Prima funzione = %f", somma_quadrati(frac, 1, 1000));
printf("Seconda Funzione =%f", somma_quadrati(sin,2,13));
}

double frac(double x){
return 1.0/x;
}
```

# Funzioni come parametri:
# array di puntatori a funzioni

si consideri il seguente codice

```
typedef double (*pfdd)(double, double)     /* definisco pfdd come un nuovo tipo
                                               puntatore funzione che riceve in input
                                               due double e  restituisce un double */
/* definisco tre funzioni che matchano  con il tipo del puntatore definito
   precedentemente */
double f1 (double, double);
double f2(double, double);
double f3(double, double);

double funzione(pfdd f,  int, int);   /* una funzione con una funzione come  parametro */

pfdd   f[4] ={NULL, f1,f2,f3}; /* un array di puntatori a funzioni inizializzato */

double x = funzione(f[1], 54, 56);                    /* due esempi di espressioni */
double y = f[2](43.2, funzione(f[2], 3.0,4.0));
```

# Qualificatori di tipo

**Const e volatile**

il qualificatore const accanto ad una variabile seganla che quella variabile non può essere modificata in seguito nel programma
si consideri il seguente codice

```
const int n=3;

int v[n]                          /* vietato  il compilatore segnala problemi */

int *p= &n;                       /* vietato: problemi su ++*p */
const int *p= &n;                 /*  p è un punt. a una costante int: ammesso */

int *const p = &n;                /* p è un puntatore costantead un intero: ammesso
                                  Non è ammesso assegnare valori a p ma si a *p */
const int *const p = &n        /* p è un puntatore costante a un intero costante: ammesso
                                  non è possibbile modificare ne p ne *p */
```

**volatile** un oggetto volatile può  essere modificato dall'HW in modo non specificato