array e for

LEGGERE

capitolo 5 del testo

l'array è una struttura dati che permette di mettere assieme molti oggetti dello stesso tipo facilitandone l'accesso

char X[100]; dichiara 100 variabili carattere che si chiamano:

X[0], X[1], ……, X[99]

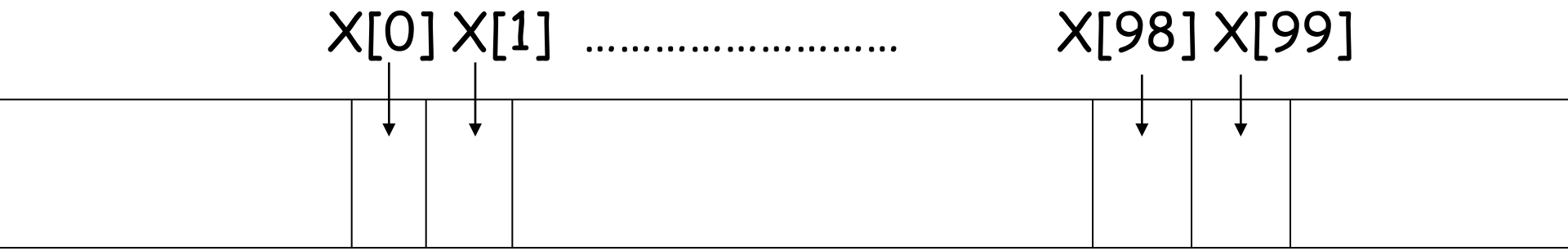
attenzione : si parte sempre dall'elemento 0

il C++ PRESCRIVE che questi elementi abbiano SEMPRE L-valori contigui

X[11] e X[12] allora

&X[12] = &X[11] +1 ricordarsi che ogni char occupa 1 byte

e quindi &X[99] = &X[0]+99



RAM

lo stesso per array di qualsiasi tipo usando il n. di byte del TIPO

array a 2, 3, 4, 5, …. dimensioni:

double X[5][10];

float Y[3][4][10];

int Z[10][10][20][30];

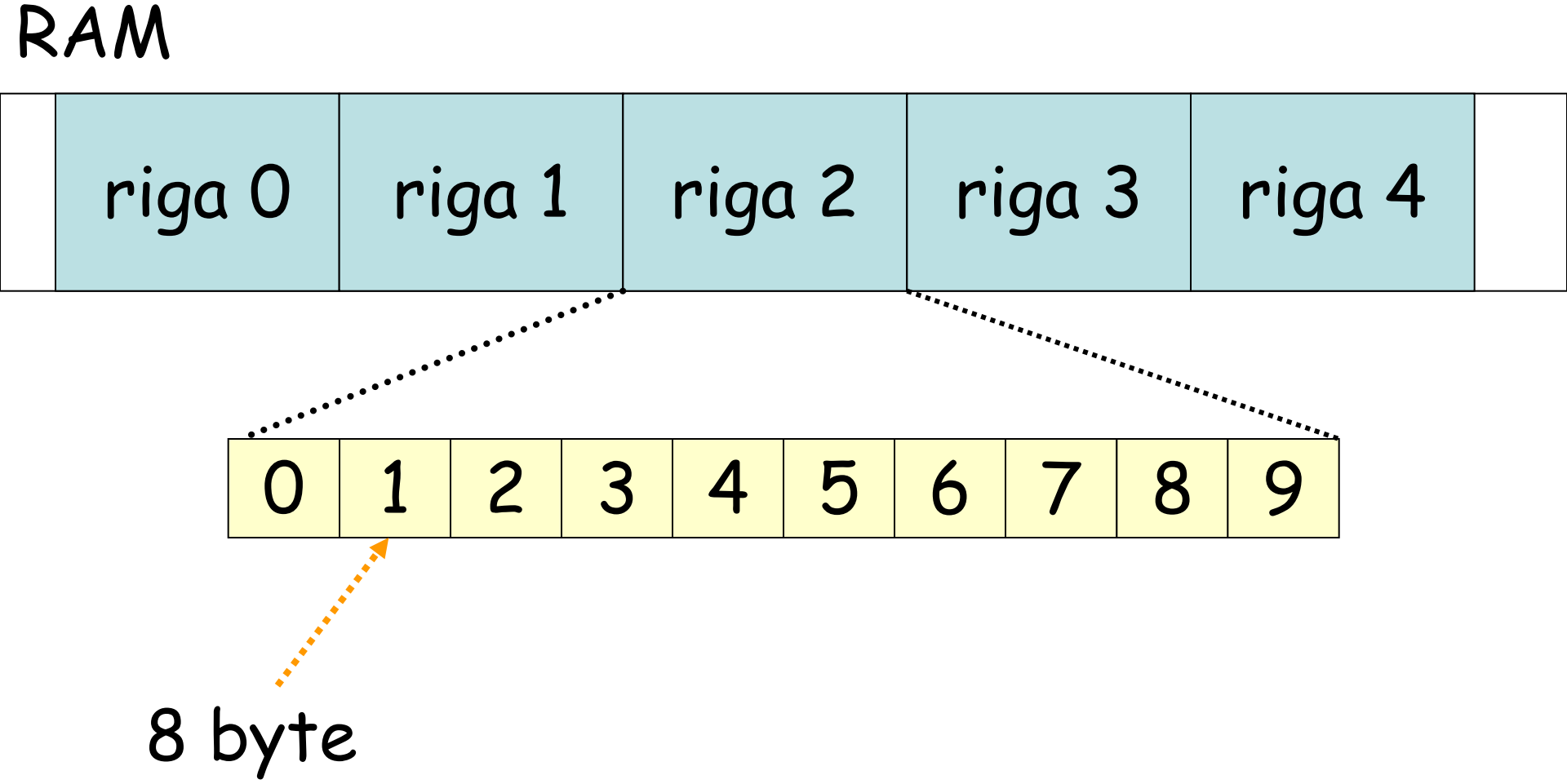
e così via

limite della prima dimensione è 5, della seconda è 10

elementi: X[0][0] X[4][9]
Z[0][0][0][1]

Y[3][0][1] non esiste

di nuovo gli elementi sono accostati nella RAM per righe: double X[5][10]



e float Y[3][4][10]; ?

ogni strato è un array [4][10] visto prima

il for viene usato spesso con gli array

for (Ini ; Test ; Incr)

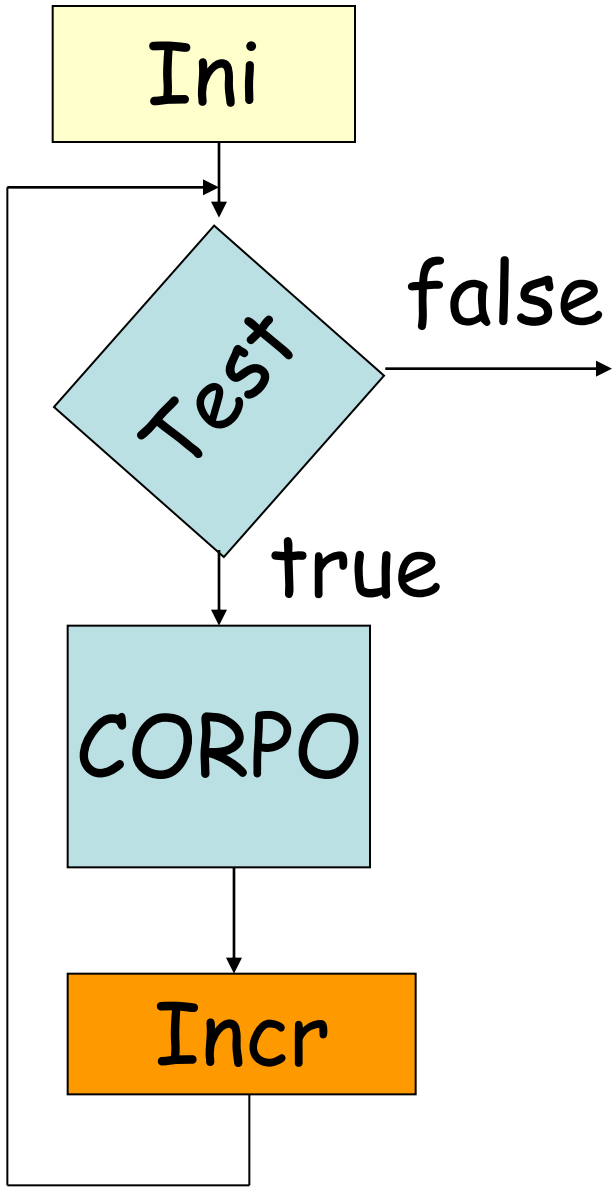
{ CORPO}

```
int A[100], somma=0;

for(int i=0; i<100; i++)
somma=somma+A[i];
```

attenzione a i !

```
int Z[10][10][20][30], somma=0;

for(int i=0; i<10; i++)
      for(int j=0; j<10; j++)
                  for(int k=0; k<20; k++)
                        for(int w=0; w<30; w++)
                         somma=somma+Z[i][j][k][w];
```

```
for ( Ini ; Test ; Incr)
{ CORPO}
```



```
for ( Ini ; Test ; Incr)
{ CORPO}
```

è equivalente a

```
Ini;
while (Test)
{CORPO; Incr;}
```

in certi casi è più comodo il for in altri il while

che tipo ha un array ?

```
char A[100];

cout<< A<< &A[0];   // stampa 2 valori uguali
```

A è un puntatore al primo elemento ed è anche costante.

provate a compilare `A=A+1; // da errore`

PERCHE'? se cambiassi A perderei l'accesso all'array e questo non può essere GIUSTO !!

char A[100];

Il tipo esatto di A è char[100],

ma in tutti i contesti char[100] è equivalente a char[] e a char *

e anche a char * const

OSSERVARE

```
char A[100], *p =A; // OK char[100] è
                    //quasi un puntatore a char

char B[100];

B=p;  // ERRORE B è costante
```

perché ?

char[100] è convertito in char *

B è costante !

```
int A[100], *p=A;
```

A[20] e p[20] sono esattamente la stessa cosa

o anche *(p+20)

quindi p[20] è lo stesso di *(p+20)

la cosa pericolosa è che :

```
int *q; q[30]=1;   // è considerato OK
                   // dal compilatore
```

eventualmente errore RUN TIME

inizializzare un array

```
int Z[10][10][20][30], somma=0;

for(int i=0; i<10; i++)
      for(int j=0; j<10; j++)
                  for(int k=0; k<20; k++)
                        for(int w=0; w<30; w++)
                              IN >> Z[i][j][k][w];
```

```
int Z[10][10][20][30]={0,…,59999}
```

altre inizializzazioni

```
int X[10]={};  // tutti 0

char Y[5]={'p','i','p','p','o'};
```

ARITMETICA dei puntatori in C++:

- dato T * p;
- p+1 == p + sizeof(T)
- *p = oggetto puntato da p
-*(p+1) = oggetto puntato da p+1
-*(p+k)= p[k]

anche se p non è un array

char * p; p+5 ?

double *q, q-10 ?

double P[100]; P+4?

insomma sapere il tipo puntato e quindi la sua dimensione è essenziale

sembra facile, ma si complica con gli array a più dimensioni

e che tipo hanno gli array a più dimensioni?

int K[5][10]; tipo = int (*) [10] = int [][10]

char R[4][6][8]; tipo = char (*) [6][8] = char[][6][8]

double F[3][5][7][9]; tipo = double (*)[5][7][9]

..... e cosa otteniamo se:

cout<< K << R << F ; ?

&K[0][0] &R[0][0][0] &F[0][0][0][0]

double F[3][5][7][9]; tipo = double (*)[5][7][9]

tipo di *F ? e R-valore di *F ?

tipo di **F ? e R-valore di **F ?

tipo di ***F ? e R-valore di ***F ?

tipo di ****F ? e R-valore di ****F ?

CAPIRE

double F[3][5][7][9];

allora ***F ha tipo double * e

***(F+4) ha anch'esso tipo double*

la dereferenziazione cambia il tipo,

ma l'aritmetica cambia il valore non il tipo !!!

CAPIRE :

float K[3][5][7][10];

K[1]= *(K+1)

K[3][2]=*(*(K+3)+2)

K[2][1][4][1]= *(*(*(*(K+2)+1)+4)+1)

esercizio:

float*** K[3][5][7][10];

-tipo di K, di *K di *(K+3) di *(*(K+3)-2)

-tipo di K[0] di K[-1][-2] e di K[3][1][0]

-valori ?

float K[3][5][7][10];

se K == L, che valore ha K[2][2]?

K[2] = L + 2*(5*7*10) *4 =2800 =L1
K[2][2] = L1 + 2*(7*10)*4 = 560

K[3][5][10] = ?

float K[3][5][7][10]; se K=L

K[-1][-2] = ?

K[-1]-K = L - (5*7*10)*4 = L1

K[-1][-2] = L1 - 2*(7*10)*4

K[-1][-2][5] = L -(5*7*10)*4 - 2*(7*10)*4 + 5*10*4

Gli array di char si comportano diversamente dagli array di altri tipi

1) Inizializzazione:

```
int A[]={0,1,2,3,4,56,99};//ha 7 elementi

char B[]={'p','i','p','p','o'};// ha 5 elementi

char C[]="pippo";   //ha 6 elementi
```

C[5] contiene '\0' carattere nullo
codice ASCII = 0

2) STAMPA

il carattere nullo serve da sentinella che segnala la fine stringa e serve per molte operazioni sulle stringhe

infatti cout<<"pippo";

si comporta allo stesso modo di:

cout << C; // stampa pippo

C[3]='\0' ; cout << C; ??

e cosa stampa:

```
char B[]={'p','i','p','p','o'};
cout<< B;
```

???