# tipo degli array

5.3 del testo, pag. 65

# che tipo ha un array ?

char A[100]; cout<< A<< &A[0]; // stampa 2 indirizzi uguali. Quindi A ha tipo char* ?

Il tipo esatto di A è char[100]

ma in C++ char[100] è (quasi) lo stesso di char * (e anche di char[])

ma non sempre : sizeof(char*)=>4

sizeof(char[100])=>100

# A è una costante

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

PERCHE'?

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

ma A+1 è espressione valida che indica il puntatore all'elemento di indice 1 dell'array, A+k punta all'elemento di indice k

# OSSERVARE

```
char A[100],  *p =A; // OK
char B[100];
B=p;  // ERRORE B è costante
```

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

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

la cosa pericolosa è che :

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

eventualmente ci sarà errore RUN TIME

scambiare 2 array:

```
int A[20], B[20];
scambia(A,B, 20); // A e B scambiati

void scambia(int X[], int*Y, int dim)
{
 int t;
 for(int i=0; i<dim; i++)
   {t=X[i]; X[i]=Y[i]; Y[i]=t;}
}
```

Potrei invocare scambia anche così:

scambia(A, &A[10], 10);

e questo è lo stesso di scrivere:

scambia(A, A+10, 10);

aritmetica dei puntatori: idea molto semplice

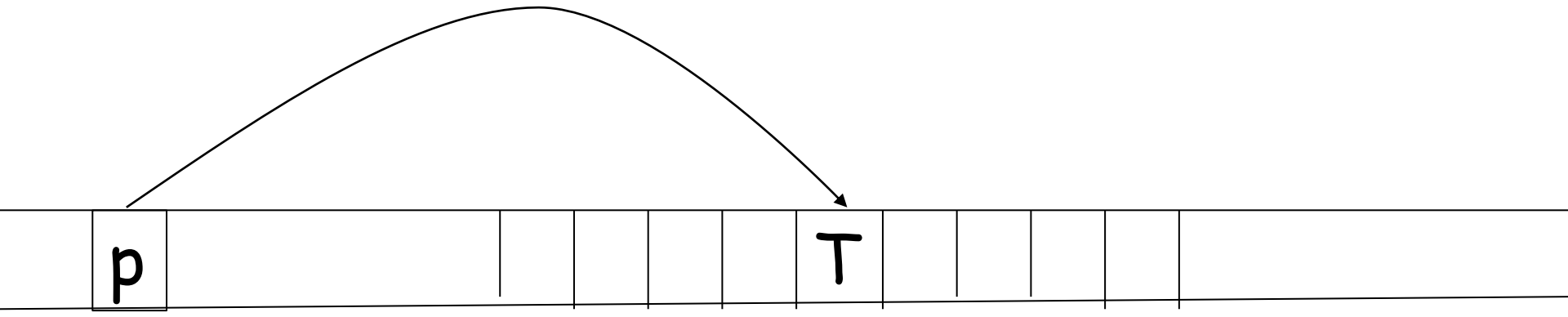
# ARITMETICA dei puntatori in C++:

- double * p; int * q;
- p+1 == p + 1*8 ha tipo double*
- q+5 = q+5*4 ha tipo int*
-*(p+1) = p[1]= oggetto puntato da p+1
-*(q+5)=q[5]= oggetto puntato da q+5

anche se p e q non sono array !!!

➔il tipo del puntatore è importante

# come il C++ vede un puntatore di tipo T*



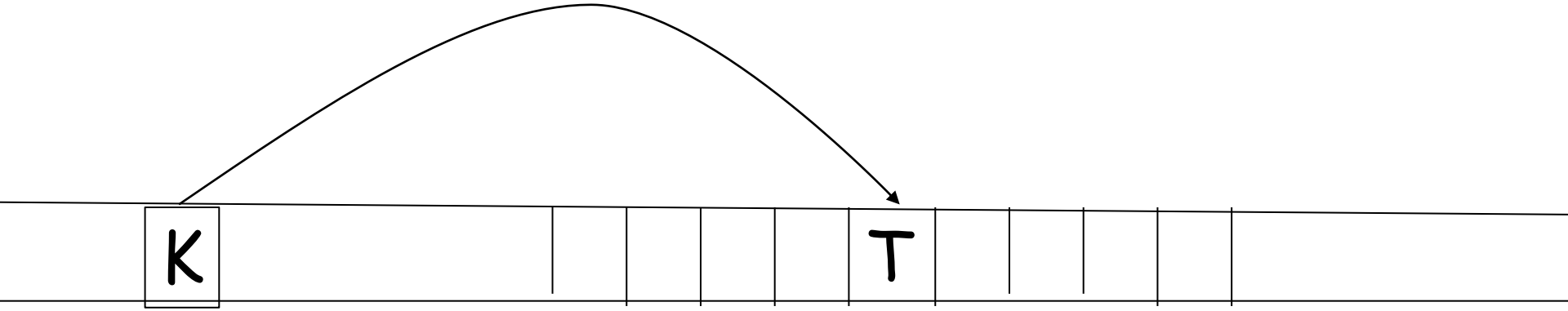

p punta ad un elemento di un array di elementi di tipo T, quindi p+n sposta il puntatore sugli elementi a destra e p-n su quelli a sinistra

# tipo e dimensione degli oggetti puntati

int K[5][10]; tipo di K = int (*) [10] = int [][10]
dimensione=10*4

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

double K[3][5][7][9]; tipo = double (*)[5][7][9]
dimensione=8*5*7*9



ma per ogni K,

cout<<K ;

produce la stampa dell'indirizzo (L-valore) del primo elemento dell'array

attenzione:

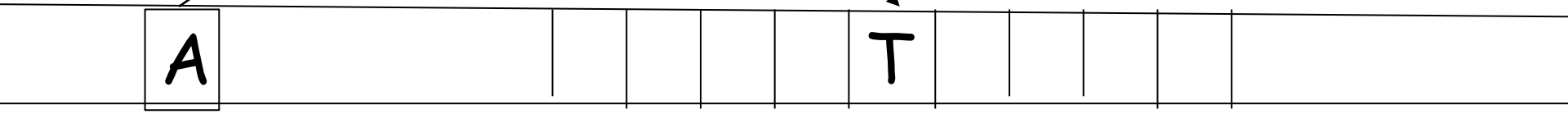
```
int A[3][5][7][9];

 cout<<sizeof(int(*)[5][7][9])<<' '<<sizeof(A)<<'
'<< sizeof(int [5][7][9])<<endl;
```

stampa 4  945*4 e 315*4

dim. di tutto A

dim. elemento base



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

tipo di *F e R-valore di *F ?
double (*)[7][9] e &F[0][0][0][0]

tipo di **F e R-valore di **F ?
double (*)[9] e &F[0][0][0][0]

tipo di ***F e R-valore di ***F ?
double * e &F[0][0][0][0]

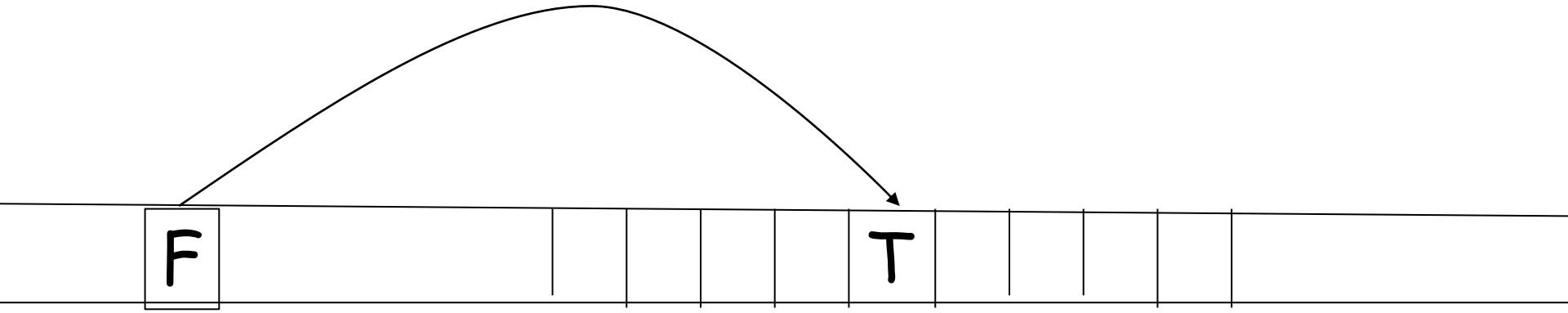
tipo di ****F ? e R-valore di ****F ?
double e valore di F[0][0][0][0]

F, *F, **F, ***F sono tutti puntatori, ad oggetti di dimesioni diverse e quindi a loro si applica l'aritmetica dei puntatori con effetti diversi

che valore ha F+2 e (*F)+2 e (**F)+2 e (***F)+2 ?

basta sapere il tipo di ciascun puntatore

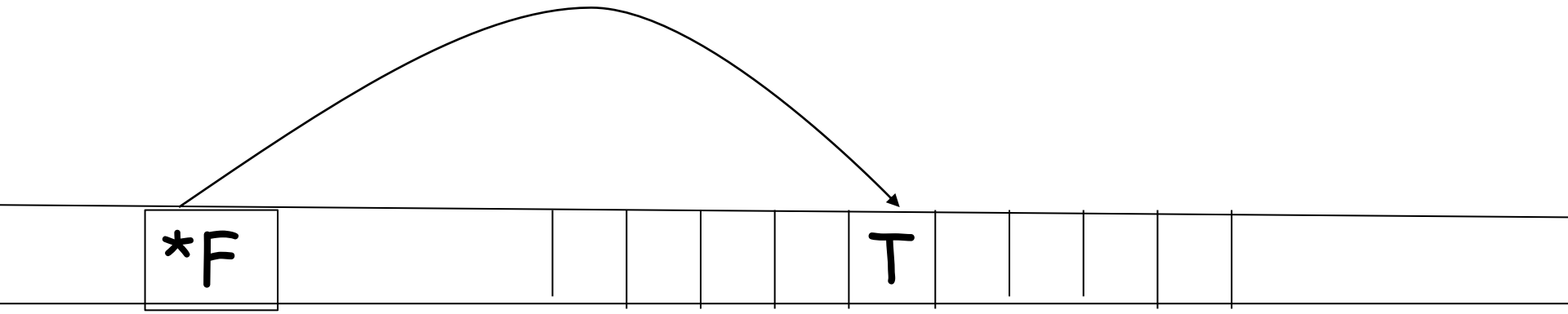
partiamo da F ha tipo double (*) [5][7][9]
T ha dimensione = (5*7*9)*8=315*8




F +1 = F+315*8
F+2=F+2*315*8
F-5 =F-5*315*8
eccetera

tutti valori di tipo
double (*)[5][7][9]

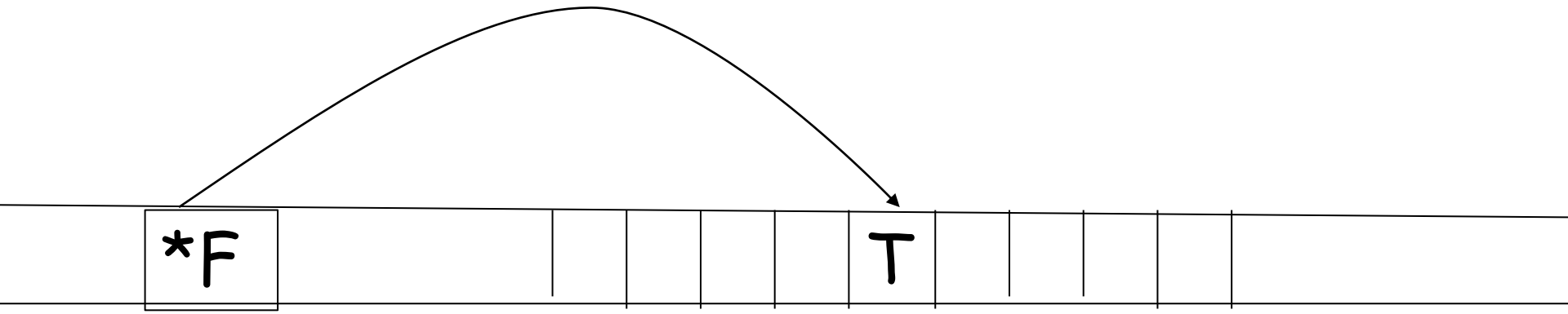
*F ha tipo double (*) [7][9]
T ha dimensione = (7*9)*8=63*8




*F +1 = F+63*8
*F+2=F+2*63*8
*F-5 =F-5*63*8
eccetera

tutti valori di tipo
double (*)[7][9]

**F ha tipo double (*) [9]
T ha dimensione = 9*8=72




*F +1 = F+72
*F+2=F+2*72
*F-5 =F-5*72
eccetera

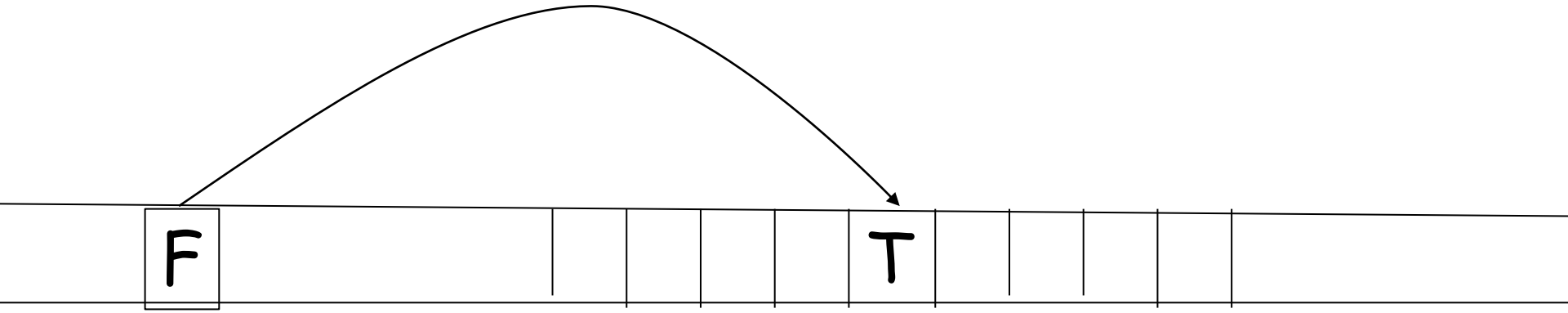
tutti valori di tipo
double (*)[9]

e subscripting ?

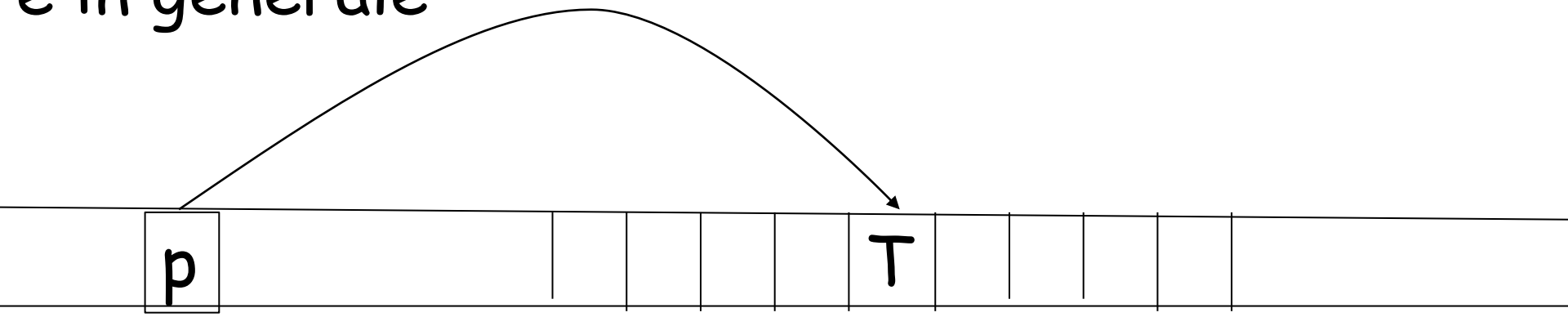
double F[3][5][7][9]; tipo = double (*)[5][7][9]

F[3] = *(F+3)  F[-2]=*(F-2)

F ha tipo double (*) [5][7][9]
T ha dimensione = (5*7*9)*8=315*8



e in generale




p[k] =*(p+k*T)

attenzione: nel [k] c'è sempre la dereferenziazione *

# 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];

indicando con K il valore di K, che valore ha K[2][2]?

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

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

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

K[-1][-2] = ?

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

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

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

tutti gli elementi degli array sono contigui nella RAM, dato: int Z[4][5][6][10];

Gli elementi di Z partono da &Z[0][0][0][0] e sono 4*5*6*10. Se vogliamo sommarli:

```
int * prossimo=&Z[0][0][0][0], somma=0;
for(int n=0; n<4*5*6*10; n++)
somma=somma + prossimo[n];
```

invariante?
R=( 0<=n<=10*10*20*30) &&
somma= prossimo[0..n-1])

# 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 !!!

passare array a funzioni:

float K[3][5][7][10]; F(K); //OK

allora

F(float (*X)[5][7][10])

ma se vogliamo che F "lavori" su
float A[10][5][7][10] // no problem!
ma con float B[10][5][8][10] // non va !

possiamo passare qualsiasi array ad una funzione come fosse ad 1 dimensione e poi possiamo calcolare "a mano" gli indici di strati, righe e colonne che ci servono.