passare array a funzioni

testo 7.3 pag. 89

abbiamo visto che se vogliamo passare un valore int/double/... a una funzione allora la funzione deve avere un parametro formale int/double/...

e per passare un array?

dobbiamo conoscere il tipo degli array

iniziamo con array ad una dimensione

float X[10];

X è una costante e ha tipo

float * const

o float const []

il valore di X è &X[0]

X è costante: X=X+1; da errore di compilazione

funzione : void F(float *z) {…} (o F(float z[]))

chiamante:

float C[20]; F(C);

float B[10]; F(B)

float a =2.3f, *p=&a; F(p);

a) sempre passaggio per valore,
b) array di float con qualsiasi numero di elementi
c) in C un puntatore è considerato come un array
d) C e B sono costanti e z non lo è, funziona per (a)

quindi possiamo passare a F indifferentemente:

1) il nome di un array float di qualunque numero di elementi

2) un puntatore a float

void F(float *z){ z=z+2; ….} //ok

passaggio per valore ➔ non ci sono side effects

```
//esempio PRE={A[0..dim_A-1] definito}
int max(int*A, int dim_A)
{ int max=A[0];
   for(int i=1; i< dim_A; i++)
   if(max < A[i])
         max=A[i];
   return max;
} //POST={max è massimo di A[0..dim_A-1]}
main()
{int K[400]; .....; int m = max(K,400);.....}
```

```
int pippo[10]={……}, pluto[20]={……};

int max_pippo=max(pippo,10);

int max_pluto=max (pluto,20);
```

max accetta array di interi con diverso numero di elementi

10, 20, 30,….10000,…..

2 cose da capire

1) i tipi int[10] e int[20] sono lo stesso tipo ➔ int* o int[]

se non fosse così dovremmo avere tanti max specializzati per ogni dim. di array

max10(int[10],…), max20(int[20],…) e così via

INACCETTABILE

il primo PASCAL (’70) era così !

2) in f(int * z,…) viene passato per valore solo il puntatore al primo elemento dell'array e NON una copia dell'array

c'è un SOLO array: quello del chiamante

la funzione può cambiare l'array,

se non vogliamo:

F(const int *A,…)

```
void F(int *A)
{A[0]=A[1];}
main()
{
int x[]={0,1,2,3,4};
F(x);
cout<<x[0]<<endl;  // ?
}
```

e ?

```
void F(int *A)
{A++; A[1]++;}
```

c’è solo l’array x, in F, A punta a x[0]

NOTARE:

```
void F(int * & z){.....}
main()
{int x[100]={};
  F(x);   // errore di compilazione
}
```

error: invalid initialization of non const reference of type ‘int*&’ from an R-value of type ‘int*’

funzionerebbe con F(int* const & z) ma F non potrebbe cambiare z

passare ad una funzione anche array a più dimensioni

Per farlo dobbiamo sapere il loro tipo

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

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

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

un parametro formale capace di ricevere l'array

int K[5][10]; è

F(int (*A)[10]) o F(int A[][10])

F riceve con successo anche

int B[10][10] e C[20][10], ma non int D[5][11]

insomma solo il limite della prima dimensione è qualsiasi, mentre quello della seconda dimensione è FISSO

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

la riceviamo con:

…F(char (*A)[6][8]) o F(char A[][6][8])

di nuovo solo il limite della prima dimensione è libero, mentre le altre dimensioni hanno limiti fissati

OSSERVA: per array int ad una dimensione

una stessa funzione può ricevere ogni array int ad una dimensione con un qualsiasi numero di elementi

è comodo !!

ma per array a più dimensioni NON è così:

la funzione che accetta K[5][10], accetta anche K[10][10], ma non K[5][11].

perché è così?

se nel corpo di F(int A[][10]) appare l'espressione A[3][5] che richiede di accedere a quell'elemento di A

il compilatore ha A =&A[0][0] e deve calcolare l'indirizzo di A[3][5]

A +(3 righe di 10 int) + 5 int

insomma [10] nel tipo di A serve al compilatore, A[][] non gli basterebbe

ma allora si rischia :

f(int A[][10], int righe)

f1(int A[][11], int righe)

f2(int A[][12], int righe)

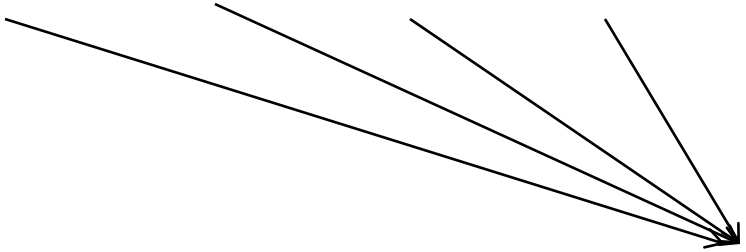
f3(int A[][13], int righe)

…… e così via?

NO !!

usiamo l'allocazione contigua degli array in memoria per trattarli tutti come array ad una dimensione

10 11 12 13 .....



```
void f(int * p, int righe, int colonne)
{

        *(p + 3*colonne + 5 )   ≡ A[3][5]

} insomma si fanno i conti “a mano”
```

se voglio “vedere” int X[1000] come B[ns][nr][nc]

```
int get3(int*A,int i,int j, int k, int ns, int nr, int nc)
{
  if  (0<=i && i<ns && 0<=j && j<nr && 0<=k   &&
                k<nc)
       return *(A+(i*nr*nc)+(j*nc)+k);
   return *A;
}
```

per esempio, se vedo X come B[5][10][10] e voglio B[1][5][8], basta invocare:

int z = get3(X,1,5,8,5,10,10);

gli array a 1 dimensione di char sono speciali

testo 5.5 pag. 69

Gli array ad 1 dimensione di char si comportano diversamente dagli array di altri tipi

1) 2 inizializzazioni:

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

char C[]=“pippo”; //ha 6 elementi

C[5] contiene ‘\0’ carattere nullo del codice ASCII = 0

2) stampa

int A[10]={1,2….};
cout << A; stampa l'R valore di A, &A[0]

char C[]=«pippo»;
cout<< C; stampa «pippo»

e il carattere '\0' è lo stop

e invece con :

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

che succede ???