



## Tipi strutturati: Array Parte 2

Ing. Nadia Ranaldo  
Dipartimento di Ingegneria  
Università degli Studi del Sannio

## Passare un array come parametro



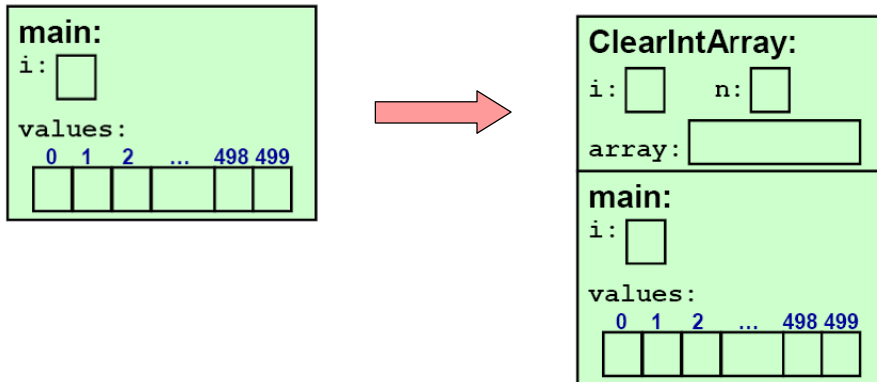
- In questo caso non avviene la copia dei dati contenuti nell'array in una variabile locale della funzione
- Viene copiato solo l'indirizzo base dell'array

```
#define N_VALUES 500
void clearIntArray(int array[], int n);
main() {
    int i;
    int values[N_VALUES];
    clearIntegerArray(values, N_VALUES);
}

void clearIntArray(int array[], int n) {
    int i;
    for (i = 0; i < n; i++)
        array[i] = 0;
}
```

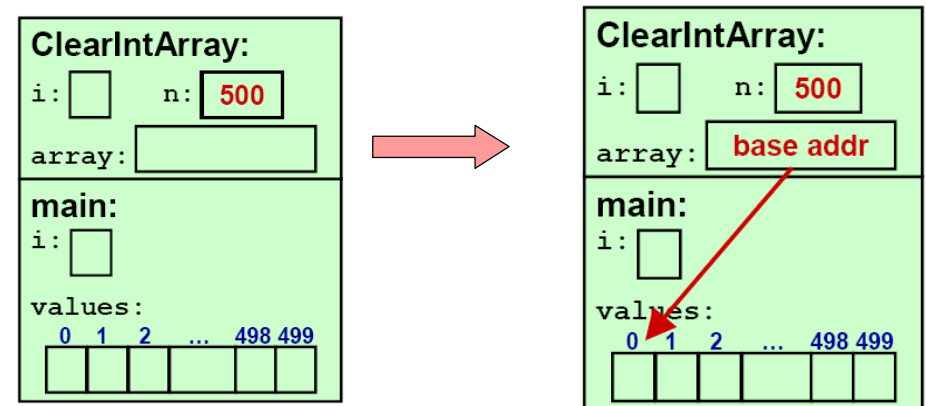
2

## Cosa accade in memoria? (1)



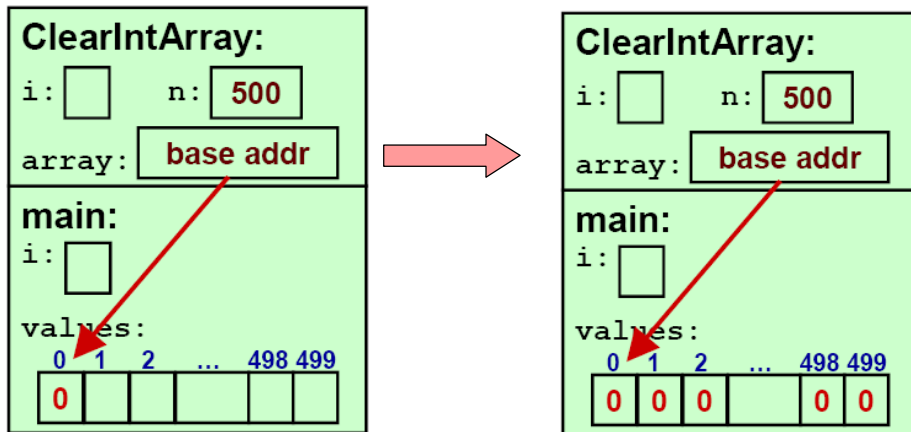
3

## Cosa accade in memoria? (2)



4

## Cosa accade in memoria? (3)

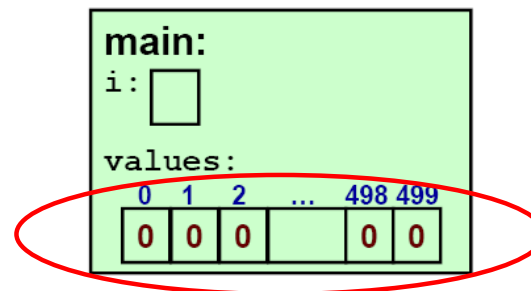


5

## Passare un array come parametro



- In questo caso le modifiche effettuate sull'array dalla funzione sono effettuate **sull'array della funzione chiamante** (del main)!
- Quando la funzione termina viene deallocato il suo stack frame ma gli effetti sull'array rimangono



6

## Capacità e riempimento (1)



- Le funzioni che ricevono un array come parametro in cui memorizzare delle informazioni devono:
  - conoscere la capacità dell'array
  - restituire il numero di informazioni memorizzate, ovvero il riempimento
- Le funzioni che ricevono un array come parametro per leggere dei dati ed effettuare delle elaborazioni devono:
  - Conoscere il riempimento dell'array (in modo da lavorare sul sottoinsieme dell'array effettivamente utilizzato)

7

## Capacità e riempimento (2)



- Esempio: scrivere un programma che legge un insieme di numeri interi in un array e ne effettua la media. Utilizzare funzioni generalizzate rispetto al numero degli elementi che può contenere l'array.
  - una funzione che inizializza gli elementi di un array mediante numeri letti da tastiera (supponiamo di leggere tanti numeri dalla tastiera quanto vale la capacità dell'array)  
`void inputArray(int a[], int capacity)`
  - una funzione che effettua la media degli elementi (si considera il riempimento dell'array)  
`float media(int a[], int riempimento)`

8

## Capacità e riempimento (3)



```

/* esegue la lettura di massimo c interi dove c è la capacità
dell'array */
void inputArray(int a[], int c){
    int i;
    for (i=0; i<c; i++) {
        printf("dammi l'el. %d-esimo di a\n", i);
        scanf("%d", &a[i]);
    }
}
/* esegue la media di un numero di elementi che può essere
inferiore alla capacità dell'array */
float media(int a[], int r){
    float media=0;
    int i;
    for (i=0; i<r; i++) {
        media+=a[i]; }
    media/=r;
    return media;
}

```

9

## Capacità e riempimento (4)

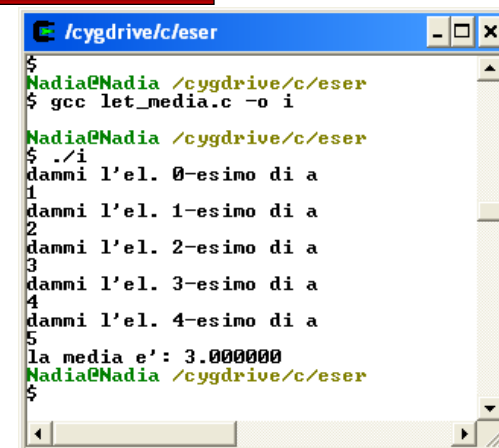


```

#include<stdio.h>
#define N 5
void inputArray(int a[], int c);
float media(int a[], int r);

main() {
    int a[N];
    inputArray(a,N);
    printf("la media e': %f", media(a,N));
}

```



```

/cygdrive/c/eser
$
Nadia@Nadia /cygdrive/c/eser
$ gcc let_media.c -o i
Nadia@Nadia /cygdrive/c/eser
$ ./i
dammi l'el. 0-esimo di a
1
dammi l'el. 1-esimo di a
2
dammi l'el. 2-esimo di a
3
dammi l'el. 3-esimo di a
4
dammi l'el. 4-esimo di a
5
la media e': 3.000000
Nadia@Nadia /cygdrive/c/eser
$

```

10

## Capacità e riempimento (5)



```

/* esegue la lettura di massimo c interi dove c è la capacità
dell'array. L'utente indica quando vuole terminare inserendo
il valore di sentinella. La funzione restituisce il numero
di elementi effettivamente letti */
int inputArray(int a[], int c, int sentinella){
    int i=0;
    int value;
    while(i<c) {
        printf("dammi l'el. %d-esimo (terminare con %d)\n", i,
sentinella);
        scanf("%d", &value);
        if (value==sentinella) break;
        a[i]=value;
        i++;
    }
    return i;
}

```



```

/cygdrive/c/eser
Nadia@Nadia /cygdrive/c/eser
$ gcc let_media2.c -o i
Nadia@Nadia /cygdrive/c/eser
$ ./i
dammi l'el. 0-esimo (terminare con 0)
1
dammi l'el. 1-esimo (terminare con 0)
2
dammi l'el. 2-esimo (terminare con 0)
3
dammi l'el. 3-esimo (terminare con 0)
0
la media e': 2.000000
Nadia@Nadia /cygdrive/c/eser
$

```

## Capacità e riempimento (6)

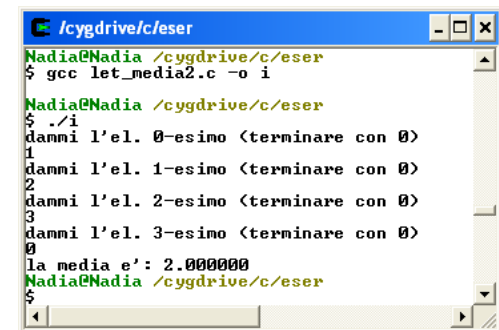


```

#include<stdio.h>
#define N 5
int inputArray(int a[], int c,int s);
float media(int a[], int r);

main() {
    int a[N],r;
    r=inputArray(a,N,0);
    printf("la media e': %f", media(a,r));
}

```



```

/cygdrive/c/eser
Nadia@Nadia /cygdrive/c/eser
$ gcc let_media2.c -o i
Nadia@Nadia /cygdrive/c/eser
$ ./i
dammi l'el. 0-esimo (terminare con 0)
1
dammi l'el. 1-esimo (terminare con 0)
2
dammi l'el. 2-esimo (terminare con 0)
3
dammi l'el. 3-esimo (terminare con 0)
0
la media e': 2.000000
Nadia@Nadia /cygdrive/c/eser
$

```

# Esempio: Creare un istogramma



- Dato un insieme di punteggi di un test (da 0 a 100)
    - Creare un diagramma che indica quanti punteggi ricadono tra 0-9, 10-19, .. 90-99, 100
- |     |           |
|-----|-----------|
| 98  | 0         |
| 84  | 10        |
| 77  | 20 *      |
| 93  | 30 *****  |
| 90  | 40 *~~~~~ |
| 85  | 50 ~~~~~~ |
| 62  | 60 ~~~~~~ |
| 99  | 70 ~~~~~~ |
| 100 | 80 ~~~~~~ |
| 87  | 90 ~~~~~~ |
| ... | 100 **    |

# Soluzione (1)



- Utilizziamo un array `counts` di 11 elementi, in cui l'elemento `i`-esimo viene usato per contare i punteggi compresi nell'intervallo  $[i*10, (i+1)*10-1]$
- Ad esempio `i=0` intervallo `[0,9]`, `i=3` intervallo `[30,39]`, `i=10` intervallo `[100,109]`

|        |    |  |
|--------|----|--|
| counts | 0  |  |
|        | 1  |  |
|        | 2  |  |
|        | 3  |  |
|        | 4  |  |
|        | 5  |  |
|        | 6  |  |
|        | 7  |  |
|        | 8  |  |
|        | 9  |  |
|        | 10 |  |

# Soluzione (2)



|        |    |    |           |
|--------|----|----|-----------|
| counts | 0  | 0  | 0         |
|        | 1  | 0  | 10        |
|        | 2  | 1  | 20 *      |
|        | 3  | 5  | 30 *****  |
|        | 4  | 6  | 40 *~~~~~ |
|        | 5  | 17 | 50 ~~~~~~ |
|        | 6  | 20 | 60 ~~~~~~ |
|        | 7  | 26 | 70 ~~~~~~ |
|        | 8  | 32 | 80 ~~~~~~ |
|        | 9  | 24 | 90 ~~~~~~ |
|        | 10 | 2  | 100 **    |

# Soluzione (3)



- Struttura del programma:
  - `main`
  - Legge i punteggi nell'array `scores`
  - chiama la funzione `histogram`
- `histogram`
  - Inizializza l'array `counts`
  - Distribuisce i punteggi nell'array `counts`
  - Visualizza l'istogramma

## Soluzione (4)



```
#define MAX_EXAMS 500 /*capacità dell'array scores*/
#define MAX_SCORE 100
#define MIN_SCORE 0
#define N_RANGES 11
void histogram(int scores[], int nScores);
main(){
    int scores[MAX_EXAMS];
    int nScores; /* riempimento */
    /*Legge i punteggi e setta il valore di nScores */
    histogram(scores, nScores);
}
void histogram(int scores[], int nScores){
    int counts[N_RANGES];
    . . .
```

Viene passato sia l'array che il suo riempimento

17

## Soluzione (5)



### histogram

- inizializza l'array invocando la funzione `clearIntArray`
- distribuisce i punteggi invocando la funzione `distributeScores`
- visualizza l'istogramma invocando la funzione `displayHistogram`

```
#define MAX_EXAMS 500
#define MIN_SCORE 0
#define MAX_SCORE 100
#define N_RANGES 11

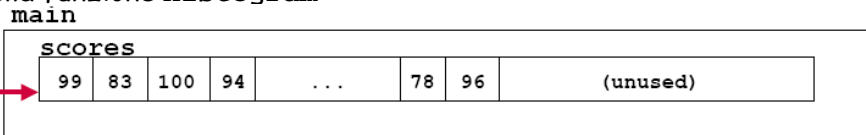
void histogram(int scores[], int nScores) {
    int counts[N_RANGES];
    clearIntArray(counts, N_RANGES);
    distributeScores(scores, nScores, counts);
    displayHistogram(counts);
}
```

18

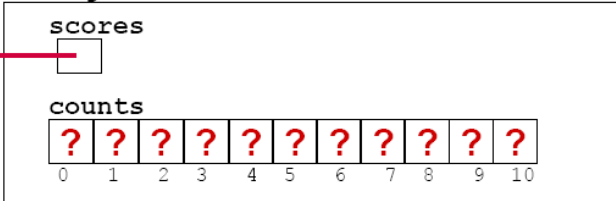
## Soluzione (5)



- Stato della memoria all'atto dell'invocazione della funzione `histogram`
- l'indirizzo di base dell'array `scores` viene copiato nella variabile locale della funzione `histogram`
- il valore di `nScores` del `main` viene copiato nella variabile `nScores` della funzione `histogram`



### Histogram



19

## Soluzione (6)



- Funzione `clearIntArray`

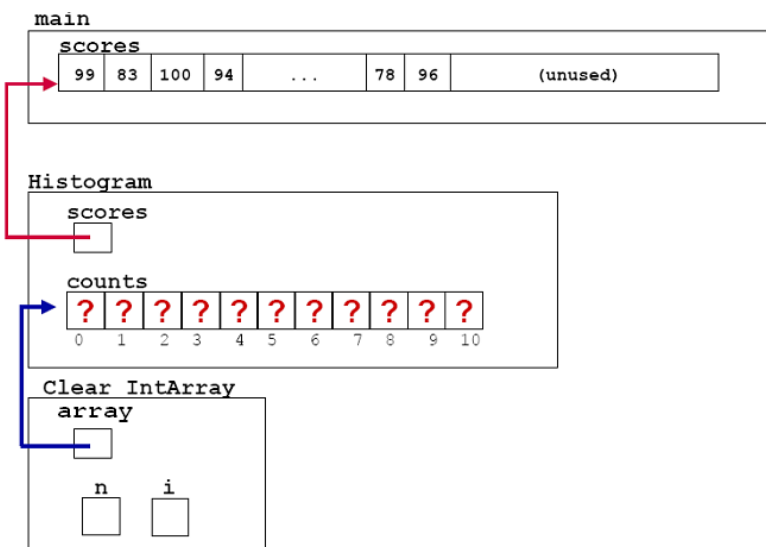
```
void clearIntArray(int array[], int n) {
    int i;
    for (i = 0; i < n; i++)
        array[i] = 0;
}
```

20

## Soluzione (7)



- Stato della memoria all'atto dell'invocazione della funzione `clearIntArray`

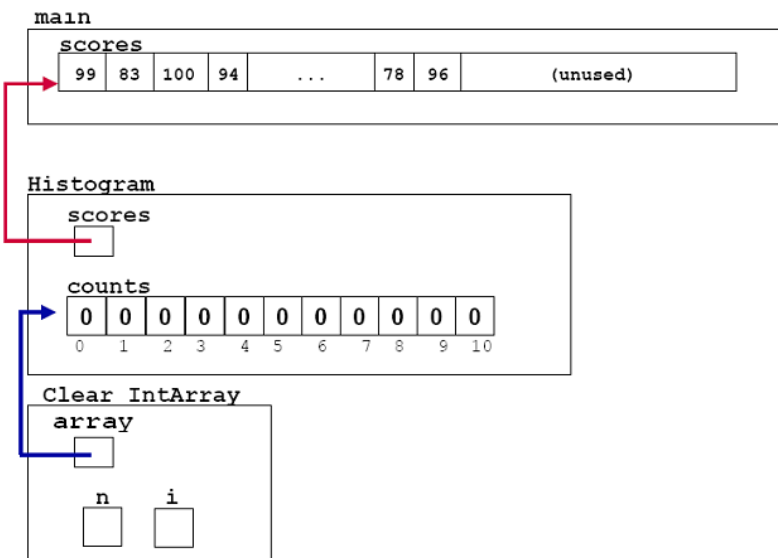


21

## Soluzione (8)



- Stato della memoria dopo l'esecuzione della funzione `clearIntArray`

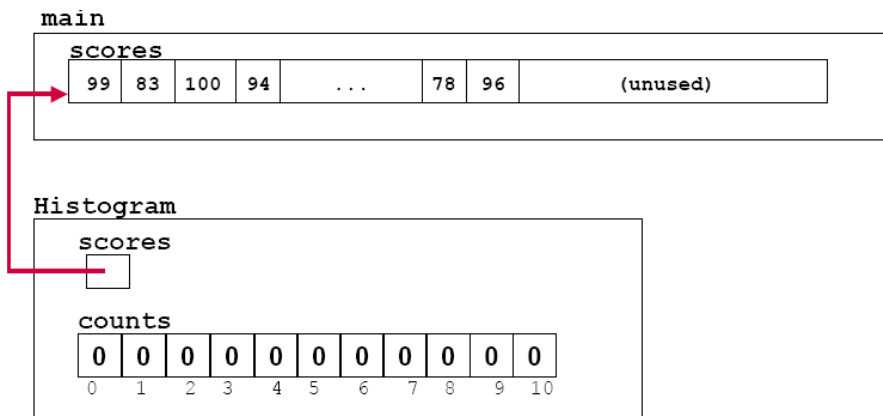


22

## Soluzione (9)



- Stato della memoria quando termina la funzione `clearIntArray`



23

## Soluzione (10)



- Funzione `distributeScores`

```
#define MAX_EXAMS 500
#define MIN_SCORE 0
#define MAX_SCORE 100
#define N_RANGES 11

void distributeScores(int scores[], int nScores,
                    int counts[N_RANGES]){

    int i, range;
    for (i = 0; i < nScores; i++) {
        range = scores[i] / 10;
        counts[range]++;
    }
}
```

• Es.  $34/10 = 3$ ;  $2/10 = 0$ ;  $100/10 = 10$ ;

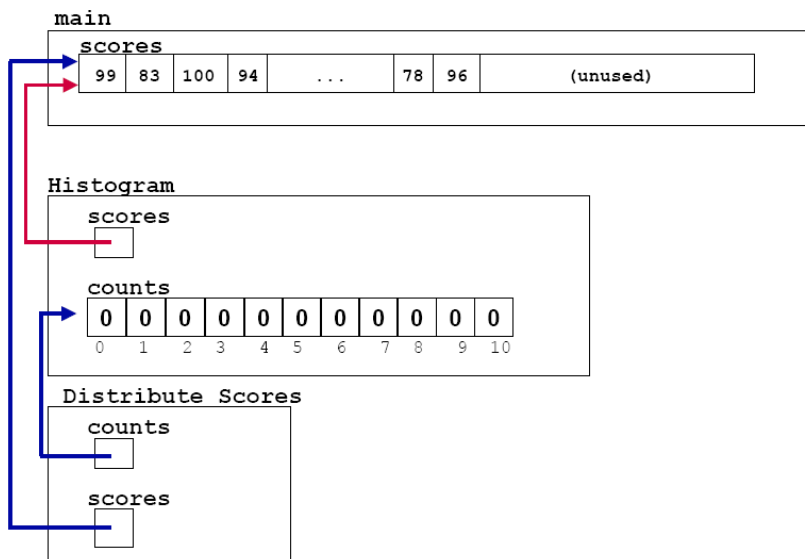
• quindi un punteggio di 34 fa aumentare di 1 il valore di `counts[3]`, un punteggio di 2 fa aumentare di 1 il valore di `counts[0]`, etc.

24

## Soluzione (11)



- Stato della memoria all'atto dell'invocazione della funzione `distributeScores`

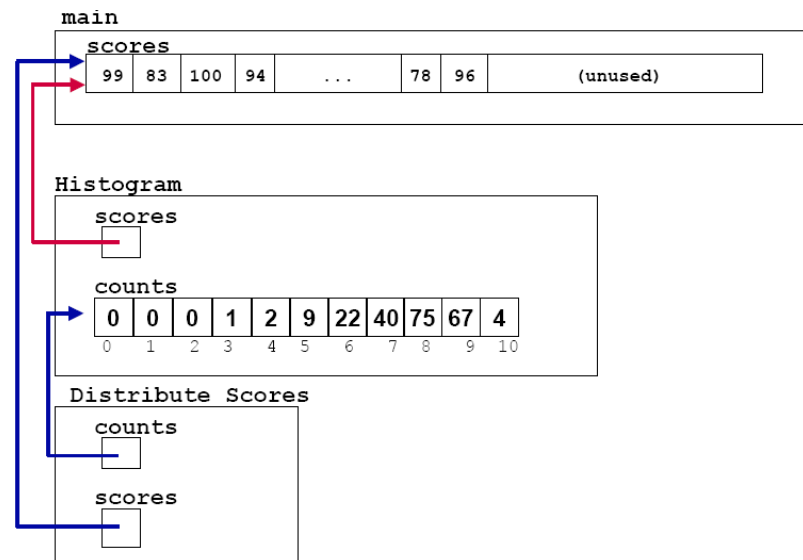


25

## Soluzione (12)



- Stato della memoria dopo l'esecuzione della funzione `distributeScores`

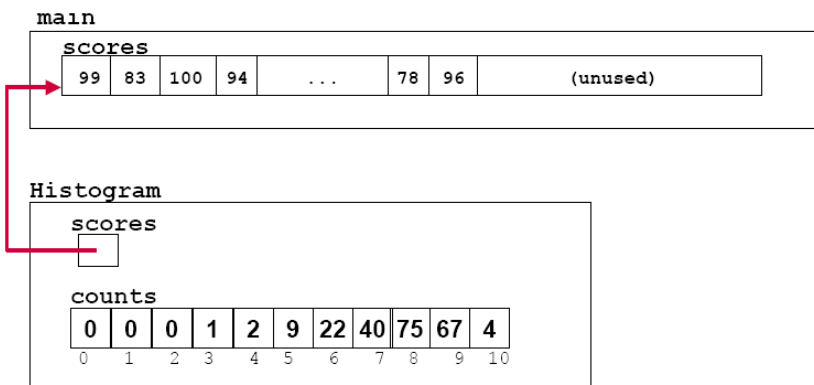


26

## Soluzione (13)



- Stato della memoria quando termina la funzione `distributeScores`



27

## Soluzione (14)



- Funzione `displayHistogram`

```
#define MAX_EXAMS 500
#define MIN_SCORE 0
#define MAX_SCORE 100
#define N_RANGES 11

void displayHistogram(int counts[N_RANGES]){
    int i,j;
    for (i = 0; i < N_RANGES ; i++) {
        for (j = 0; j < counts[i] ; j++)
            printf("*");
        printf("\n");
    }
}
```

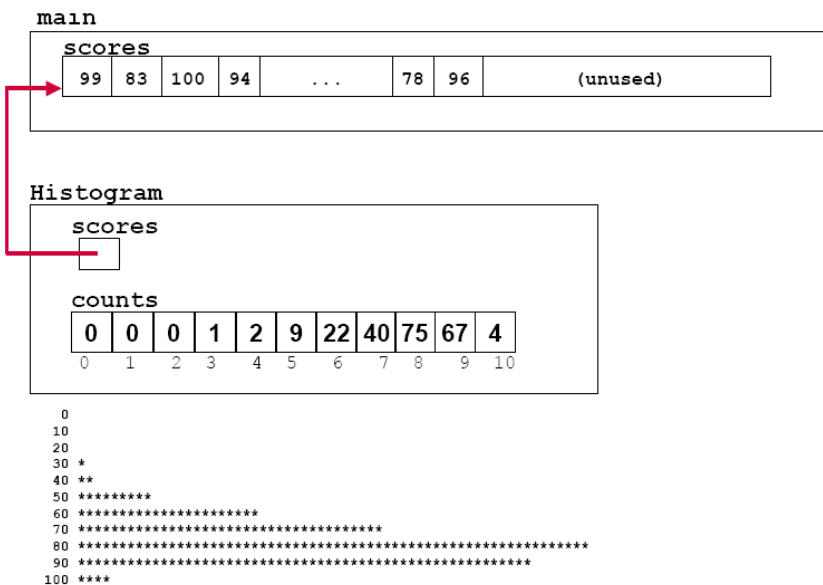
- stampa tanti asterischi quanto è il valore di `counts[i]`

28

## Soluzione (15)



- Stato della memoria quando termina la funzione `displayHistogram`



29

## Soluzione (16)



- `istogr.c`

```

/cygdrive/c/eser
Nadia@Nadia /cygdrive/c/eser
$ gcc istogr.c -o is
Nadia@Nadia /cygdrive/c/eser
$ ./is
14 3 4 2 13 24 36 76 54 56 34 23 87 67 5 64 88 23 56 87
0 0 0 0 0 0 0 0 0 0
4 2 3 2 0 3 2 1 3 0 0
****
**
***
**
****
**
*
***
Nadia@Nadia /cygdrive/c/eser
$
  
```

30

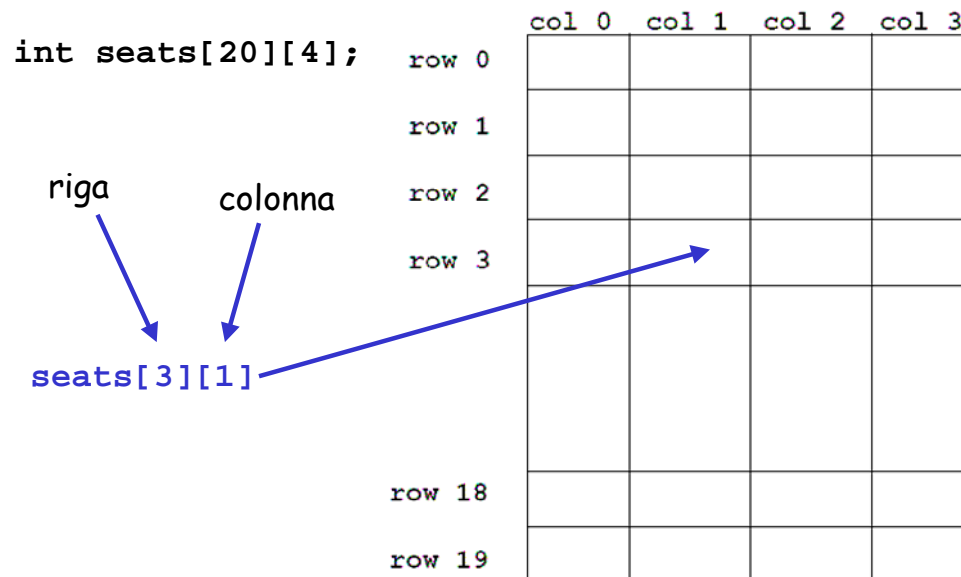
## Array multidimensionali



- In C gli elementi di una array possono essere di un qualsiasi tipo
- Quindi gli elementi di un array possono essere essi stessi degli array
- Un array di array è chiamato array multidimensionale
- Il più comune è l'array **bidimensionale** (o matrice)
  - Utilizzato spesso per rappresentare dati organizzati in una struttura rettangolare in cui ciascun elemento è individuato da una **riga** e da una **colonna**
  - Il primo indice rappresenta la riga, il secondo rappresenta la colonna

31

## Array bidimensionale



32

## Inizializzazione



- E' possibile inizializzare un array bidimensionale in maniera analoga agli array monodimensionali, specificando le dimensioni (opzionale la prima dimensione)

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

```
1 2
3 4
```

```
int b[][3]={1,2,3,4,5,6,7,8,9};
```

```
1 2 3
4 5 6
7 8 9
```

- E' possibile anche utilizzare le parentesi graffe per maggiore chiarezza

```
int b[][3]={{1,2,3},{4,5,6},{7,8,9}};
```

33

## Array multidimensionali: passaggio a funzione (1)



- E' possibile passare un array multi-dimensionale ad una funzione
- Il C richiede di specificare la dimensione di ciascun indice nell'array, tranne del primo (opzionale)
- Ad esempio consideriamo una funzione che inizializza a zero tutti gli elementi della matrice `seats`
  - Poiché la dimensione del primo indice (ovvero il numero delle righe) è opzionale, è possibile scrivere una funzione parametrizzata rispetto ad esso

```
void initSeats(int arr[][4], int nRows) {
    int row, col;
    for (row = 0; row < nRows; row++)
        for (col = 0; col < 4; col++)
            arr[row][col] = 0;
}
```

35

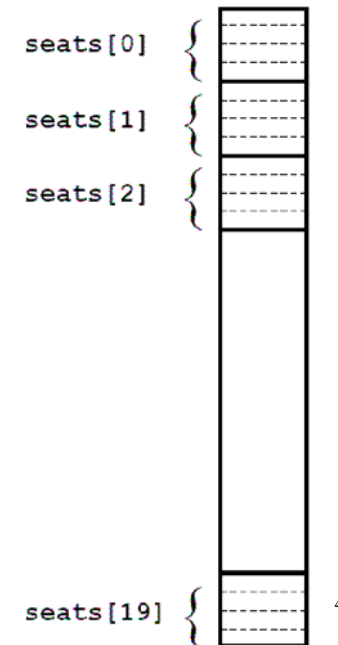
## Array bidimensionali: rappresentazione in memoria



- In memoria gli elementi di un array bidimensionale sono memorizzati come una lista monodimensionale, in particolare per riga

- Per l'array `seats`, prima i 4 elementi della riga di indice 0 (con indice di colonna che varia da 0 a 3), poi i 4 elementi della riga di indice 1, (con indice colonna che varia da 0 a 3), etc.

- In generale per un array multidimensionale `a` il primo indice varia meno rapidamente degli altri, ovvero gli elementi di `a[0]` sono memorizzati prima degli elementi di `a[1]`, e così via per gli altri indici.



## Array multi-dimensionali: passaggio di parametro a funzione (2)



```
#define NROWS 20
#define NCOLS 4
void initSeats(seats, NROWS);
main() {
    int seats[NROWS][NCOLS];
    initSeats(seats, NROWS);
}

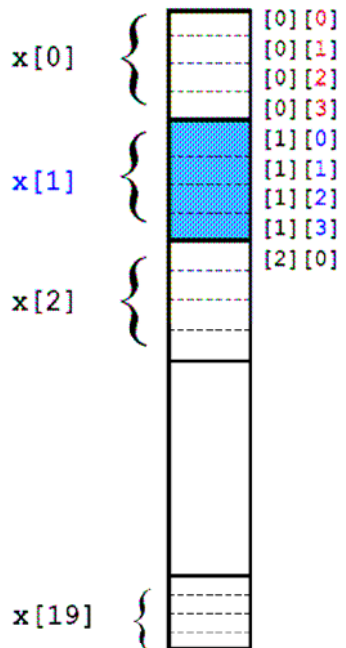
void initSeats(int arr[][NCOLS], int nRows){
    int row, col;
    for (row = 0; row < nRows; row++)
        for (col = 0; col < NCOLS; col++)
            arr[row][col] = 0;
}
```

36

## Array multi-dimensionali: passaggio di parametro a funzione (3)



- Dato un array bidimensionale a, l'elemento i-esimo a[i] è un array monodimensionale
- In generale, dato un array multi-dimensionale a di n dimensioni, l'elemento i-esimo a[i] è un array di (n-1) dimensioni
- Esempio di funzione che calcola la somma degli elementi di un array monodimensionale utilizzando un elemento di un array bidimensionale



```
int x[20][4];
int result;
result = somma(x[1], 4);
...
int somma(int arr[], n) {
    int i, sum = 0;
    for (i = 0; i < n; i++)
        sum += arr[i];
    return sum;
}
```

## Esercizio (2)



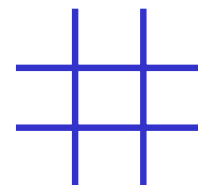
|             |             |             |
|-------------|-------------|-------------|
| board[0][0] | board[0][1] | board[0][2] |
| board[1][0] | board[1][1] | board[1][2] |
| board[2][0] | board[2][1] | board[2][2] |

```
void displayBoard(char board[3][3]);
main() {
    char a[][]={{'X','X','X'},{'O','O','X'},{'X','O','O'}};
    printf("\n");
    displayBoard(a);
}
```

## Esercizio (1)



- Scrivere una funzione per rappresentare il gioco del tris
  - Si utilizza uno schema di tre righe e tre colonne inizialmente vuoto



- I giocatori a turno inseriscono una lettera (X o O) in un quadrato vuoto
- Lo scopo è cercare di inserire tre lettere uguali orizzontalmente, diagonalmente o verticalmente
- Per rappresentare lo schema è possibile utilizzare un array bidimensionale di dimensione tre per tre
  - per semplicità un array di char che può contenere ' ', 'X' oppure 'O'

```
char board[3][3];
```

## Esercizio (3)



```
void displayBoard(char board[3][3]){
    int row, col;
    for (row = 0; row < 3; row++){
        if(row!=0)
            printf("----+\n");
        for (col = 0; col < 3; col++) {
            if(col!=0)
                printf("|");
            printf(" %c ", board[row][col]);
        }
        printf("\n");
    }
}
```

```
~/cygdrive/c/eser
X | O | O
+---+
O | O | X
+---+
X | O | O
Nadia@Nadia /cygdrive/c/eser
$ gcc tris.c -o i
Nadia@Nadia /cygdrive/c/eser
$ ./i
X | X | X
+---+
O | O | X
+---+
X | O | O
Nadia@Nadia /cygdrive/c/eser
$
```

## Esercizio (4)



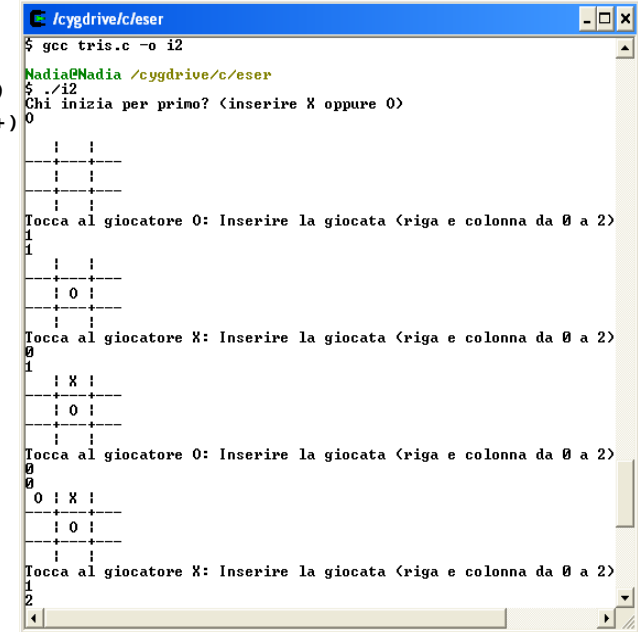
```
#include<stdio.h>
void init(char a[3][3]);
void displayBoard(char board[3][3]);
main() {
    int turno, x, y;
    char primo, a[3][3];
    init(a);
    printf("Chi inizia per primo? (inserire X oppure O)\n");
    scanf("%c",&primo);
    printf("\n");
    displayBoard(a);
    for(turno=1; turno<=9; turno++) {
        printf("Tocca al giocatore %c: ", primo);
        printf("Inserire la giocata (riga e colonna da 0 a 2)\n");
        scanf("%d", &x);
        scanf("%d", &y);
        a[x][y]=primo;
        displayBoard(a);
        primo= (primo=='X')? 'O': 'X';
    }
}
```

41

## Esercizio (5)



```
void init(char a[3][3]){
    int row, col;
    for (row=0; row<3; row++)
        for (col=0; col<3; col++)
            a[row][col]=' ';
}
```



```
cygdrive/c/eser
$ gcc tris.c -o i2
Nadia@Nadia /cygdrive/c/eser
$ ./12
Chi inizia per primo? <inserire X oppure O>
0
  | | |
--|---
  | | |
  | | |
Tocca al giocatore 0: Inserire la giocata <riga e colonna da 0 a 2>
1
1
  | | |
--|---
  | 0 |
  | | |
Tocca al giocatore X: Inserire la giocata <riga e colonna da 0 a 2>
0
1
  | X |
--|---
  | 0 |
  | | |
Tocca al giocatore 0: Inserire la giocata <riga e colonna da 0 a 2>
0
0
0 | X |
--|---
  | 0 |
  | | |
Tocca al giocatore X: Inserire la giocata <riga e colonna da 0 a 2>
1
2
  | | |
--|---
  | | |
  | | |
```