

# Esercitazione 5

## ago di Buffon

```

/* inizializzare il seme della sequenza casuale una sola volta qua. MAI nelle funzioni */
srand48(time(0));

/* chiedere il numero di lanci da effettuare */
do{
    printf("Inserire numero di lanci 0 < N <= %d: ", NMAX);
    scanf("%d", &lanci);
} while( lanci<=0 || lanci>NMAX);

/* acquisire la distanza tra le linee */
do{
    printf("Inserire la distanza d tra le linee 0 < d <= %.1f cm: ", DMAX);
    scanf("%.1f", &d);
} while( d<=0 || d>DMAX);

/* chiedere lunghezza dell'ago. passiamo d come argomento
per poter imporre la condizione che sia L < d */
do{
    printf("Inserire la lunghezza ago 0 < L <= %.1f cm: ", d);
    scanf("%.1f", &L);
} while( L<=0 || L>d);

/* ripetere NEXP esperimenti */
for(iexp=0; iexp<NEXP; iexp++) {

    /* azzerare il numero di successi per ciascun esperimento */
    successi = 0;

    for(j=0; j< lanci; j++) {
        /* generare il valore di x */
        x = 0.5*d*lrnd48()/RAND_MAX;

        /* generare il valore di theta */
        theta = 0.5*M_PI*lrnd48()/RAND_MAX;

        /* aumentare i successi se si verifica la condizione */
        if( x < (L/2.)*sin(theta) ) {
            successi++;
        }
    } /* fine lanci */

    /* calcolo pi_greco */
    pigreco[iexp] = (2.*L*lanci)/(successi*d);
} /* fine esperimenti */

```

$i = 1; i \leq N-1$   
 $i = 1; i < N; i++$   
 $i = 1, \dots, i = 2, \dots, i = N-1$

$i = 0; i \leq N; i++$   
 $i = 0; i = 1, \dots, i = 2, \dots, i = N$   
 $N+1$  iterazioni

$i = 1; i \leq N$   
 $i = 0; i < N$

double data[N]

0	...		N-1
	✓	✓	✓

$i = 1; i \leq N; i++$

data[1] = - - ✓

data[N-1] = - - ✓

data[N] = - X

double pigreco[1000]

indici: 0, ..., 999

pigreco[0] = - -

pigreco[999] = - -

pigreco[i] = - -

$i \in [0, \dots, 999]$

```

/* dichiarare massimo, minimo e valore medio piMax, piMin, piAvg */
/* azzerare il valore medio che usiamo per sommare i valori ottenuti */
/* inizializzare il massimo con un valore piccolo, e il minimo con un valore grande */
piAvg = 0;
piMin = 10.;
piMax = 0.;

// ciclo per calcolo min, max, media
for(iexp=0; iexp<NEXP; iexp++) {

    // scrittura sul file
    fprintf(fp, "%5d \t %.12lf\n", iexp, pigreco[iexp]);

    /*aggiornare il massimo */
    if( pigreco[iexp] > piMax ) piMax = pigreco[iexp];

    /*aggiornare il minimo */
    if( pigreco[iexp] < piMin ) piMin = pigreco[iexp];

    /* aumentare la somma che poi useremo per la media */
    piAvg += pigreco[iexp];
}

/* calcolo della media DOPO aver terminato gli esperimenti */
piAvg /= NEXP;
printf("valore di pigreco massimo: %.12lf\n", piMax);
printf("valore di pigreco minimo: %.12lf\n", piMin);
printf("valore medio di pigreco: %.12lf\n", piAvg);

```

$$piMin = pigreco[0]$$

$$piMax = pigreco[0]$$

$$piAvg = piAvg / NEXP$$

```

/* dichiarare massimo, minimo e valore medio piMax, piMin, piAvg */
/* azzerare il valore medio che usiamo per sommare i valori ottenuti */
/* inizializzare il massimo con un valore piccolo, e il minimo con un valore grande */
piAvg = 0;
piMin = 10.;
piMax = 0.;

/* ripetere NEXP esperimenti */
for(iexp=0; iexp<NEXP; iexp++) {

    /* azzerare il numero di successi per ciascun esperimento */
    successi = 0;

    for(j=0; j< lanci; j++) {
        /* generare il valore di x */
        x = 0.5*d*rand48()/RAND_MAX;

        /* generare il valore di theta */
        theta = 0.5*M_PI*rand48()/RAND_MAX;

        /* aumentare i successi se si verifica la condizione */
        if( x < (L/2.)*sin(theta) ) {
            successi++;
        }
    } /* fine lanci */

    /* calcolo pi_greco */
    pi = (2.*L*lanci)/(successi*d);
    pigreco[iexp] = pi;

    // scrittura sul file
    fprintf(fp, "%-4d \t %.12lf\n", iexp, pi);

    /*aggiornare il massimo */
    if( pi > piMax ) piMax = pi;

    /*aggiornare il minimo */
    if( pi < piMin ) piMin = pi;

    /* aumentare la somma che poi useremo per la media */
    piAvg += pi;
} /* fine esperimenti */

/* calcolo della media DOPO aver terminato gli esperimenti */
piAvg /= NEXP;

```

## ordinamento di array

int data[] = {9, 1, 2, 5, 3, 7, 4};

Criteri:  
1/ da più piccolo a più grande  
2/ da più grande a più piccolo

① {1, 2, 3, 4, 5, 7, 9};

② {9, 7, 5, 4, 3, 2, 1};

data[3] = ?

originale: 5

1/ : 4

2/ : 4

= {9, 1, 2, 5, 3, 7, 4};

1, 9, 2, 5, 3, 7, 4

1, 2, 9, 5, 3, 7, 4

1, 2, 5, 9, 3, 7, 4

1, 2, 5, 3, 9, 7, 4

↓

1, 2, 5, 3, 7, 4, 9

Bubble sort

```
#define LEN 4
```

```
int main() {
```

```
    int data[LEN] = {9, 3, 4, 1};
    int i, j, temp;
```

```
    // stampa array
```

```
    printf("pre-ordinamento data = { ");
    for(i=0; i<LEN; i++) printf("%d ", *(data+i));
    printf("}\n\n");
```

```
    // ciclo i sugli elementi dall'inizio
```

```
    for(i=0; i<LEN; i++) {
```

```
        // ciclo j sugli elementi da ultimo fino a i+1
```

```
        for(j=LEN-1; j>i; j--) {
```

```
            printf("i: %d \t j: %d \t data[%d]: %d \t data[%d]: %d\n",
                   i, j, j-1, *(data+j-1), j, *(data+j));
```

```
            if(*(data+j-1) > *(data+j)) {
                printf("=> scambia data[%d] con data[%d]\n\n", j, j-1);
                temp = *(data+j);
                *(data+j) = *(data+j-1);
                *(data+j-1) = temp;
            } // fine scambio
```

```
        } // ciclo j sugli elementi successivi a i
```

```
    } // ciclo i su tutti gli elementi
```

```
    // stampa array
```

```
    printf("dopo ordinamento data = { ");
    for(i=0; i<LEN; i++) printf("%d ", *(data+i));
    printf("}\n\n");
```

```
} // fine main
```

$*(data + i) \equiv data[i]$

$i = 0, 1, 2, 3$

$i = 0$

$j = 3, 2, 1$

$i = 1$

$j = 3, 2$

$i = 2$

$j = 3$

$i = 3$

$j$  : non assume  
al cun valore

determine  
il criterio  
 $>$   
 $<$

$\{ \quad \}$   
 $i \longrightarrow \quad \longleftarrow j$

$*(data + j - 1) \equiv data[j - 1]$

$*(data + j) \equiv data[j]$

$i = 0$

$j = 3$

$j - 1 = 2$

if ( data[2] > data[3] )

$\downarrow$   
4

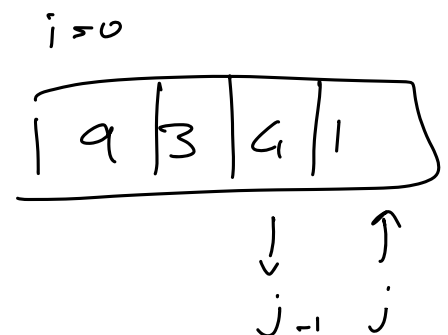
$\downarrow$   
1

$\Rightarrow$  Scambio  $j \longleftrightarrow j - 1$

temp = 1

data[3] = 4

data[2] = temp  
2

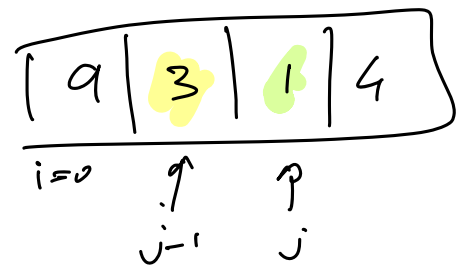


$$i = 0$$

$$j = 2 \quad j-1 = 1$$

if (data[j-1] > data[j]) ✓

Scambio

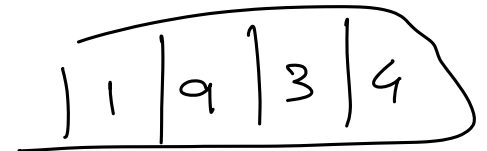
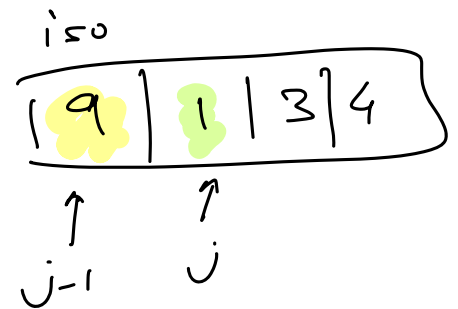


$$i = 0$$

$$j = 1 \quad j-1 = \emptyset \quad i = j-1$$

if (data[j-1] > \* (data + j)) ✓

Scambio



~~$i = 0$   
 $j = 0$~~   $j > i$

$$i = 1$$

$$j = 3 \quad j-1 = 2$$

```

[ShaBookPro14-562:material rahatlou$ gcc -o /tmp/app bubblePtr.c
[ShaBookPro14-562:material rahatlou$ /tmp/app
pre-ordinamento data = { 9 3 4 1 }

i: 0    j: 3    data[2]: 4    data[3]: 1
==> scambia data[3] con data[2]

i: 0    j: 2    data[1]: 3    data[2]: 1
==> scambia data[2] con data[1]

i: 0    j: 1    data[0]: 9    data[1]: 1
==> scambia data[1] con data[0]

i: 1    j: 3    data[2]: 3    data[3]: 4
i: 1    j: 2    data[1]: 9    data[2]: 3
==> scambia data[2] con data[1]

i: 2    j: 3    data[2]: 9    data[3]: 4
==> scambia data[3] con data[2]

dopo ordinamento data = { 1 3 4 9 }

```

1 9 | 3 | 4 | 1

val = 4

int found = 0;

for (int i=0; i < 4; i++) {

if (x(data+i) == val) {

found = 1;  
break;

}

if (found) {

printf("trovato %d\n", val);

} else {

printf("valore %d non in array\n", val);

}

1 | 3 | 7 | 9 | 18 | 25 | 39

array ordinato

val = 7 nell'array?

min < val < max

✓

min < val < max

✗

1 | 3 | 7 | 9

✗

✓

7 | 9

Ricerca binaria  
su array ordinato

```
#define LEN 7
```

```
int main() {
```

```
    int data[LEN] = {-5, -2, 7, 12, 14, 19, 23};
```

```
    int i, j, temp;
```

```
    int end= LEN-1, start=0, middle;
```

```
    int target;
```

```
    target = 7;
```

```
    printf("valore da cercare? ");
```

```
    scanf("%d", &target);
```

```
    printf("stai cercando il numero %d\n", target);
```

```
    // variabile di appoggio
```

```
    int found = 0;
```

```
    do{
```

```
        // indice elemento in mezzo
```

```
        middle = (start+end)/2;
```

```
        printf("start: %d \t middle: %d \t end: %d\n", start, middle, end);
```

```
        if(*(data+middle) == target) {  
            found = 1;
```

```
        } else if(*(data+middle) < target) {  
            start = middle + 1;
```

```
        } else {  
            end = middle - 1;  
        }
```

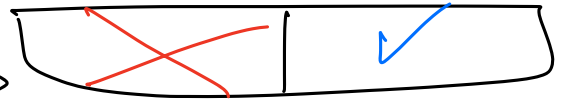
```
    } while( !found && start <= end);
```

```
    if(found) {  
        printf("trovato %d in posizione %d !\n", target, middle);
```

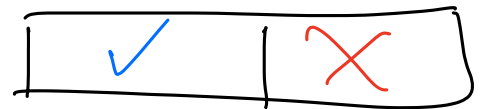
```
    } else {  
        printf("data non contiene il valore %d\n", target);  
    }
```

```
} // fine main
```

start = 0      end = 6  
middle = 3



! found  $\equiv$  found  $\neq 1$



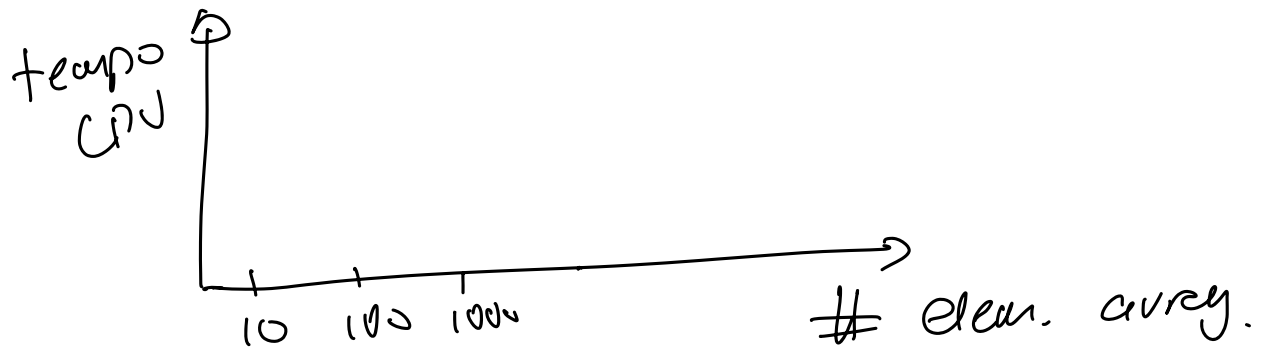
```
ShBookPro14-562:material rahatlou$ gcc -o /tmp/app binsearch.c
ShBookPro14-562:material rahatlou$ /tmp/app
valore da cercare? 5
stai cercando il numero 5
start: 0          middle:3          end: 6
start: 0          middle:1          end: 2
start: 2          middle:2          end: 2
data non contiene il valore 5
ShBookPro14-562:material rahatlou$ /tmp/app
valore da cercare? 7
stai cercando il numero 7
start: 0          middle:3          end: 6
start: 0          middle:1          end: 2
start: 2          middle:2          end: 2
trovato 7 in posizione 2 !
```

Esercizio:

- array lungo 1000
- riempire con val. int casuali
- bubble sort
- ricerca binaria con valore scelto

\$ ./programme

\$ time ./programme



$*(\text{data} + i) \equiv \text{data}[i]$

Array 2D e puntatori

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

3 righe      3 colonne

int \* p = &mat[0][1];

printf(" \*p = %d\n", \*p );

\*p = 5

mat : è un puntatore

\*mat : è un puntatore.

\*\*mat  $\equiv$  1

col=0		col=1
1	2	3
4	5	6
7	8	9

descrittore

%p

%p

%d

```
#define NROW 3
#define NCOL 3

int main() {
    int vec[NROW] = {1, 2, 3};
    int mat[NROW][NCOL] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int i, j;
```

```
// ciclo i righe
for(i = 0; i < NROW; i++) {

    // ciclo j colonne
    for(j = 0; j < NCOL; j++) {
        printf("mat[%1d][%1d] \t", i, j);
    }
    printf("\n");

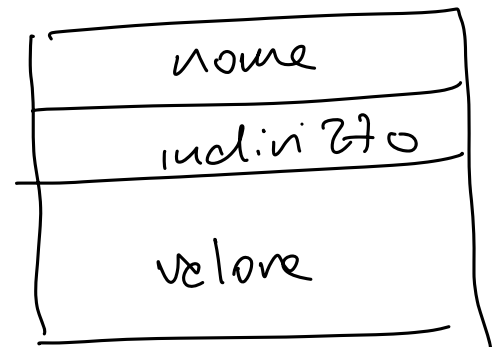
    for(j = 0; j < NCOL; j++) {
        //printf("%10p \t", *(mat+i)+j);
        printf("%10p \t", &mat[i][j]);
    }
    printf("\n");

    for(j = 0; j < NCOL; j++) {
        //printf("%10d \t", (*(mat+i)+j));
        printf("%10d \t", mat[i][j]);
    }
    printf("\n\n");
}
```

```
ShaBookPro14-562:material rahatlou$ /tmp/app
mat[0][0]      mat[0][1]      mat[0][2]
0x16b85f274      1              2              3
              1              2              3

mat[1][0]      mat[1][1]      mat[1][2]
0x16b85f280      4              5              6
              4              5              6

mat[2][0]      mat[2][1]      mat[2][2]
0x16b85f28c      7              8              9
              7              8              9
```



$mat \equiv \&mat[0][0]$

```
// mat e` puntatore -> mat[0][0]
printf("mat          : \t %p\n", mat);

// *mat puntatore -> mat[0][0]
printf("*mat (puntatore): \t %p\n", *mat);

// *mat non e` un intero
printf("*mat (intero)   : \t %d\n", *mat);

// **mat equivale a mat[0][0]
printf("**mat         : \t %d\n", **mat);
```

```
mat          :      0x16b85f274
*mat (puntatore):    0x16b85f274
*mat (intero)   :      1803940468
**mat          :          1
```

```
printf("----- mat[i] ----- \n");
```

```
// mot[i] puntatore al primo elemento della riga i
printf("mat[0] = %p\n", mat[0]);
printf("*mat[0] = %d\n", *mat[0]);
```

```
printf("mat[1] = %p\n", mat[1]);
printf("*mat[1] = %d\n", *mat[1]);
```

```
printf("mat[2]+2 = %p\n", mat[2]+2);
printf("*(mat[2]+2) = %d\n", *(mat[2]+2));
```

```
----- mat[i] -----
mat[0] = 0x16f1f3274
*mat[0] = 1
mat[1] = 0x16f1f3280
*mat[1] = 4
mat[2]+2 = 0x16f1f3294
*(mat[2]+2) = 9
```

$mat[i] \equiv \&mat[i][0]$   
inizio riga i

$$*mat[i] \equiv mat[i][0]$$

$mat[2] \rightarrow$  punta inizio riga con indice 2

$mat[2] + 2 \rightarrow$  riga = 2, colonne = 2

$$*(mat[2] + 2) \equiv mat[2][2] \quad \& \quad mat[2][2]$$

$mat[i] \rightarrow$  punta a inizio riga  $i$

$*mat$  fissa riga 0

```
printf("----- *mat+i -----\n");
```

```
// *mat+1 equivale a &mat[0][1]
printf("*mat+1 = %p\n", *mat+1);
printf("&mat[0][1] = %p\n", &mat[0][1]);
```

```
printf("*(mat+1) = %d\n", *(mat+1));
printf("mat[0][1] = %d\n", mat[0][1]);
```

```
----- *mat+i -----
*mat+1 = 0x16f1f3278
&mat[0][1] = 0x16f1f3278
*(mat+1) = 2
mat[0][1] = 2
```

```
printf("----- *(mat+i) -----\n");
```

```
// *(mat+1) equivale a &mat[1][0]
printf("*(mat+1) = %p\n", *(mat+1));
printf("&mat[1][0] = %p\n", &mat[1][0]);
```

```
printf("**mat+1) = %d\n", **mat+1));
printf("mat[1][0] = %d\n", mat[1][0]);
```

```
----- *(mat+i) -----
*(mat+1) = 0x16f1f3280
&mat[1][0] = 0x16f1f3280
**mat+1) = 4
mat[1][0] = 4
```

```
printf("----- *(mat+k)+1 -----\n");
```

```
// *(mat+2)+1 equivale a &mat[2][1]
printf("*(mat+2)+1 = %p\n", *(mat+2)+1);
printf("&mat[2][1] = %p\n", &mat[2][1]);
```

```
printf("*(mat+2)+1 = %d\n", *(mat+2)+1));
printf("mat[2][1] = %d\n", mat[2][1]);
```

```
----- *(mat+k)+1 -----
*(mat+2)+1 = 0x16f1f3290
&mat[2][1] = 0x16f1f3290
*(mat+2)+1 = 8
mat[2][1] = 8
```

$$* \underbrace{(* (mat + l)) + k}_{\text{riga } l} \equiv mat[l][k] \quad \downarrow \quad \text{colonne } k.$$

```

printf("----- mat[k] -----\n");
//
for(i = 0; i < NROW; i++) {
    printf("mat[%d]:%p \t *mat[%d]:%i \n", i, mat[i], i, *mat[i]);
}

```

```

----- mat[k] -----
mat[0]:0x16f1f3274      *mat[0]:1
mat[1]:0x16f1f3280      *mat[1]:4
mat[2]:0x16f1f328c      *mat[2]:7

```

$mat[i] \rightarrow$  puntatore inizio riga  $i$

```

// ciclo i righe
for(i = 0; i < NROW; i++) {
    printf("===== mat[%d] = %p =====\n", i, mat[i]);

    // ciclo j colonne
    for(j = 0; j < NCOL; j++) {
        printf("mat[%1d][%1d] \t", i, j);
    }
    printf("\n");

    for(j = 0; j < NCOL; j++) {
        //printf("%10p \t", *(mat+i)+j );
        printf("%10p \t", &mat[i][j] );
    }
    printf("\n");

    for(j = 0; j < NCOL; j++) {
        //printf("%10d \t", (*(mat+i)+j) );
        printf("%10d \t", mat[i][j] );
    }
    printf("\n");
}

```

```

===== mat[0] = 0x16f1f3274 =====
mat[0][0]      mat[0][1]      mat[0][2]
0x16f1f3274      0x16f1f3278      0x16f1f327c
          1              2              3
===== mat[1] = 0x16f1f3280 =====
mat[1][0]      mat[1][1]      mat[1][2]
0x16f1f3280      0x16f1f3284      0x16f1f3288
          4              5              6
===== mat[2] = 0x16f1f328c =====
mat[2][0]      mat[2][1]      mat[2][2]
0x16f1f328c      0x16f1f3290      0x16f1f3294
          7              8              9

```

$\&mat[i][j] \equiv *(mat+i)+j$

$mat[i][j] \equiv (*(mat+i)+j)$

Mat sub:k[3][3][3]