

# C

## Dichiarazione di variabili:

- Devono iniziare con una lettera
- Possono contenere cifre numeriche e il segno di sottolineatura
- Non possono contenere spazi
- Per convenzione sono scritte in lettere minuscole oppure con iniziali maiuscole tranne la prima:

*int numeroContoCorrente, attesa, appoggio;*

## Tipi semplici di dati:

INT	16 bit intero
CHAR	8 bit carattere
Float	32 bit virgola mobile
DOUBLE	64 bit virgola mobile
VOID	

## Operatori matematici :

+  
-  
\*  
/  
% → calcola il resto tra due interi

### esempi:

```
q=5/2;  
/* q=2 */
```

```
r=5%2;  
/* r=1 */
```

```
tot=2+3*4;  
/* tot=14 */
```

**Ordine di esecuzione:** (\*) (/) (%) (+) (-)

```
prod=3*-2;  
/* prod=-6 */
```

## Operatori relazionali:

```
if (peso > limite)
/* vale 0 se falso, vale 1 se vero */
```

```
if(18) {.....}
```

←  
sempre vera perché non è zero

```
if(0) {.....}
```

←  
sempre falsa

```
tropo= (litri<3);    /* troppo è una variabile logica */
```

## Operatori logici:

```
if (altezza >=180 && larghezza <=70) {.....} → AND
```

```
if (lunghezza >400 || larghezza >165) {.....} → OR
```

```
if (k!=3) {.....} o anche: if (!a) {.....} → NOT
```

## Assegnamenti:

Per aggiungere 1 alla variabile z:

z++;

Z viene prima  
usata e poi  
incrementata

++z

Z viene prima  
incrementata  
e poi usata

```
x=4;
x=z++;
/* x=4, z=5 */
```

```
x=4;
x=++z;
/* x=5 */
```

## Istruzioni di selezione:

Esistono principalmente 4 istruzioni di selezione, vediamo alcuni esempi:

### Selezione ad un ramo:

```
if (a>b)
{
.....
.....
.....
}
```

### Selezione a due rami:

```
if (a>b)
{
.....
.....
.....
} else {
.....
.....
.....
}
```

## Istruzioni iterative:

### Selezione switch:

```
switch (a)
{
case 1:
.....
.....
break;

case 2:
.....
.....
break;

case 3:
.....
.....
}
```

### Selezione switch con default:

```
switch (a)
{
case 1:
.....
.....
break;

case 2:
.....
.....
break;

default:
.....
.....
}
```

Esistono 3 istruzioni iterative; vediamo quali sono:

## **While**

In questa iterazione prima viene verificata la condizione e poi eseguite le istruzioni

```
while(k<10) {  
.....  
.....  
.....  
}
```

## **While do**

In quest'altro tipo di while al primo ciclo vengono eseguite le istruzioni o poi verificata la condizione

```
do {  
.....  
.....  
.....  
} while(k<10);
```

## **For**

In questa istruzione di iterazione vediamo nell'ordine: l'inizializzazione della variabile i, poi la condizione che ad ogni ciclo viene verificata ed in fine l'incremento della variabile i.

```
for(i=0; i<10; i++){  
.....  
.....  
.....  
}
```

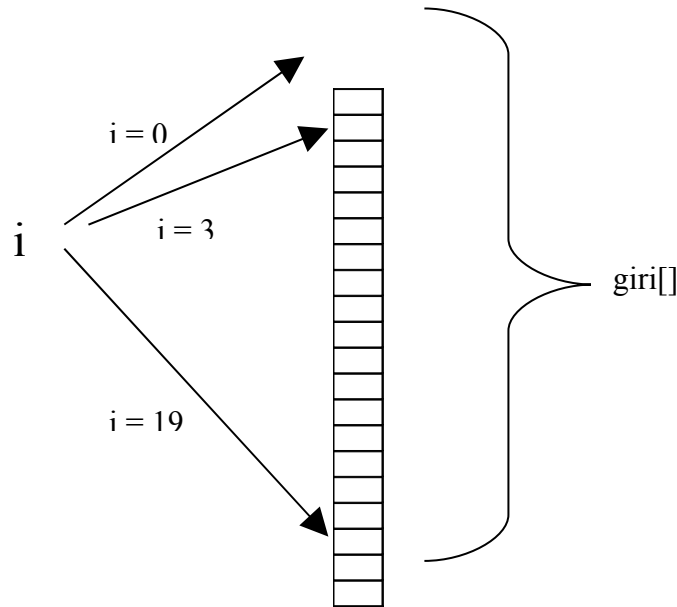
## **Array:**

Un array (o vettore) è un insieme di variabili aventi il medesimo nome e tipo (int, char, ecc.), ma distinte tra loro da un numero.

```
int giri[20]; /* 20 variabili intere */
```

L'array `giri[]` contiene 20 variabili distinte, ovvero 20 elementi.

Al fine di utilizzare l'array nel modo più facile possibile si utilizza un puntatore costituito da una variabile di tipo intero che punta sui vari elementi del vettore.



Per accedere alla casella di un vettore e copiare il suo contenuto in una variabile, si utilizza la seguente sintassi:

```
pippo=giri[i];
```

Per assegnare invece un valore in una cella del vettore:

```
giri[i]=120;
```

### Utilità degli indici nei vettori:

Grazie agli indici è possibile scorrere tutti gli elementi del vettore, all'interno di un ciclo. Ciò è molto comodo e molto utile. Vediamo un esempio:

```
int i;
char vet[3];

for (i=0; i<3; i++) {
    printf("\nImmetti un carattere nel vettore\n");
    getchar(vet[i])
}
```

nel caso sopra viene dichiarato un vettore di tipo `char` e successivamente caricato di caratteri da un operatore.

In quest'altro caso viene visualizzato a schermo il contenuto di ogni cella del vettore:

```
for(i=2; i>-1; i--) {
```

```

        printf("\n%c", vet[i]);
    }

```

**Inizializzazione di un vettore:**

I vettori così come le semplici variabili possono essere inizializzati direttamente nella dichiarazione, vediamo qualche esempio:

```
int potenze[4] = {2, 4, 8, 16};
```

Può essere inizializzato anche senza esplicitare la sua dimensione:

```
int mondiali[] = {1982, 1986, 1990, 1994};
```

**Vettori a più dimensioni:**

I vettori visti fino a questo punto sono ad una sola dimensione, ma esistono anche vettori a più dimensioni: i più usati tra questi sono le matrici (vettori a due dimensioni); vediamo un esempio di un vettore bidimensionale:

```

int codiceAutoChilometriPercorsi [9][2];

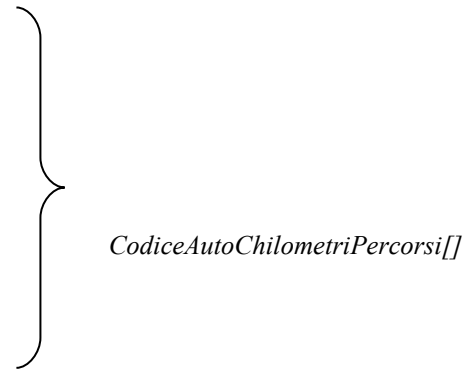
codiceAutoChilometriPercorsi [0][0] = 33;
codiceAutoChilometriPercorsi [0][1] = 121640;

codiceAutoChilometriPercorsi [1][0] = 56;
codiceAutoChilometriPercorsi [1][1] = 23122;

.....
.....

```

3	12164
3	0
5	23122
6	



**Inizializzazione di una matrice:**

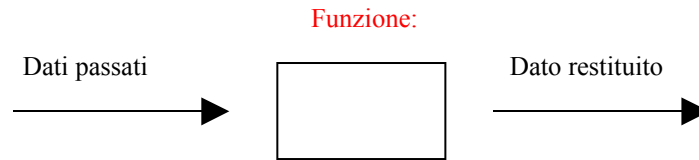
Le regole di inizializzazione di una matrice (vettore a due dimensioni) sono le stesse dei vettori monodimensionali. Vediamo un esempio:

```

int codiceAutoChilometriPercorsi [9][2] = {
    { 33, 121640},          /* prima riga */
    {56, 23122},          /*seconda riga */
    .....
    .....
    .....                /*non va messa la virgola all'ultima*/
};

```

**Funzioni:**



Le funzioni accettano valori in ingresso e restituiscono un valore in uscita:  $y = f(x)$  ad esempio in matematica dando un valore alla  $f(x)$ , appunto la  $x$ , otteniamo un valore  $y$ . In informatica le cose sono molto simili. Vediamo un esempio:

```

#include<stdio.h>
main()
{
  int som(int v[5]); /* Prototipo di funzione; la funzione va dichiarata come */
  int vet[5], i, b; /* qualsiasi altra variabile nel caso venga scritta dopo il main */

  for(i=0; i<5; i++){
    printf("\nImmetti il valore da sommare\n"); /* Viene caricato il vettore con i valori */
    scanf("%d", &vet[i]); /* da sommare */
  }

  b=som(vet); /* Viene richiamata la funzione e gli viene passato l'indirizzo del */
  printf("\n%d\n", b); /* vettore dove dovrà andare a prendere i valori da sommare */
}

/*FINE MAIN*/

int som(int v[5])
{
  int somma, i;

  somma=0;

  for(i=0; i<5; i++){
    somma=somma+v[i]; /* Viene effettuata la somma */
  }

  return somma; /* Ritorna al chiamante il valore della somma */
}
  
```

## Passaggio di parametri:

Quando viene ‘chiamata’ una funzione, gli vengono passati dei valori. Questi valori prendono il nome di: **argomento** o **parametro reale**, è il valore passato al momento della chiamata della funzione. Vediamo un esempio:

```
#include <stdio.h>

main(){
  int a, b, c;
  int funSomma(int e, int f); /* prototipo di funzione */

  printf("\nImmetti un intero\n");
  scanf("%d", &a);

  printf("\nImmetti un'altro intero\n");
  scanf("%d", &b);

  c=funSomma(a, b);

  printf("\nLa somma S: %d \n", c);
}

int funSomma(int e, int f){
  int z;

  z=e+f;
  return z;
}
```

Valore passato chiamando la funzione. In questo caso i valori sono due (*a, b*)


**Parametro o parametro formale:** variabile locale (in questo caso due variabili (*e* ed *f*)) appartenente alla funzione che riceve una copia del valore passato

Oltre agli **argomenti** o **parametri reali** abbiamo i **parametri** o **parametri formali**: sono quelli indicati nella definizione di funzione, ovvero le variabili proprie della funzione che ricevono (si caricano) i valori passati dal chiamante. Ricapitolando:

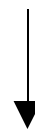
**argomento** o **parametro reale**



Il valore passato alla funzione

 Il contenuto di *b*  
`c = funSomma(a, b);`

**parametro** o **parametro formale**



La variabile presente nella funzione

 Il contenitore *f*  
`int funSomma(int e, int f){`

**Importante (passaggio per valore):**

Nell'esempio sopra riportato il passaggio dei valori alla funzione avviene per **valore**. La funzione in questione non può in nessun caso modificare il contenuto delle variabili del chiamante; riceve soltanto delle copie.

## Passaggio per riferimento:

Anche in questo caso alla funzione vengono passati delle copie, con la differenza però che le copie passate sono gli indirizzi delle variabili del chiamante, dunque la funzione può modificare a piacere il contenuto delle variabili che appartengono al chiamante. Vediamo un esempio:

```
#include <stdio.h>
```

```
void dimezza(int *pun)
{
    *pun=*pun/2;
}
```

Viene dichiarata una variabile di tipo puntatore, ovvero una variabile che contiene l'indirizzo ad un'altra variabile di tipo intero

```
main()
{
    int a;
    printf("\nImmetti un valore e otterrai il suo mezzo\n");
    scanf("%d", &a);

    dimezza(&a);

    printf("\n%d\n", a);
}
```

In questo punto viene passato alla funzione l'indirizzo della variabile *a*

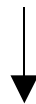
## Chiarimenti:

```
int a;
```



Viene dichiarata una variabile di tipo *int*

```
int *pun ,
```



Viene dichiarata una variabile puntatore a variabile di tipo *int*

```
*pun = *pun/2;
```

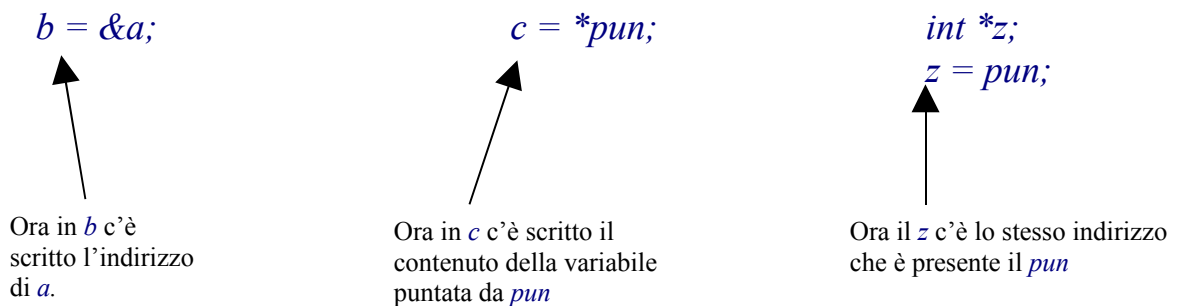
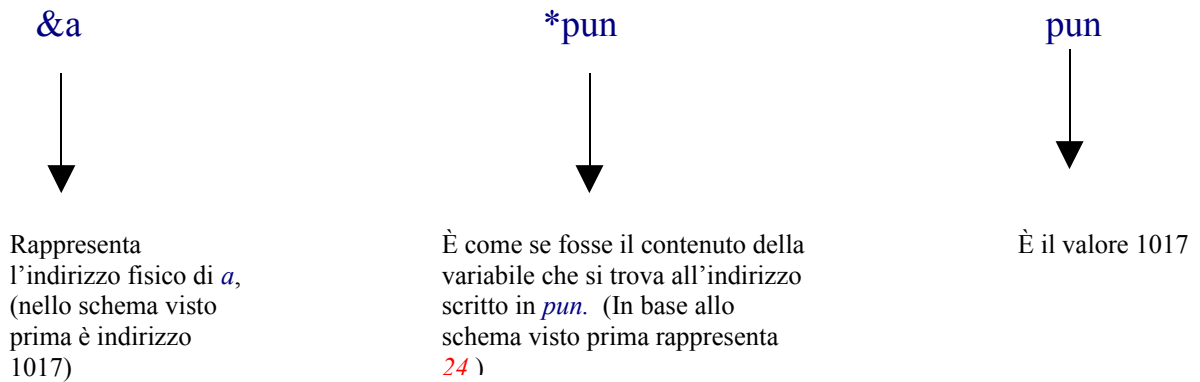


Viene modificato il contenuto della variabile il cui indirizzo si trova nella variabile *pun*

RAM	indirizzo	RAM	indirizzo	RAM	indirizzo	RAM	indirizzo	RAM	indirizzo
	1000		1005		1010		1015		1020
	1001		1006		1011		1016		1021
	1002		1007		1012	24	1017		1022
	1003	1017	1008		1013		1018		1023
	1004		1009		1014		1019		1024

La variabile *pun* contiene l'indirizzo della variabile *a*. Dunque si dice che 'punta' alla variabile *a*

La variabile *a* si trova all'indirizzo 1017



### Il caso dei vettori:

I vettori a differenza dei tipi di variabili visti fino a questo punto, vengono passati sempre per riferimento e mai per valore. O meglio ciò che viene passato è una copia dell'indirizzo del vettore in oggetto. Vediamo un esempio:

```
#include <stdio.h>

void visualizza(int v[2])
{
    printf("\n%d\n", v[0]);
    printf("\n%d\n", v[1]);
}

main()
{
    int vet[2];

    printf("\nImmetti un valore nella prima casella del vettore\n");
    scanf("%d", & vet[0]);

    printf("\nImmetti un valore nella seconda casella del vettore\n");
    scanf("%d", & vet[1]);

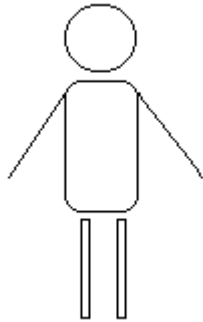
    visualizza(vet);
}
```

← La funzione che una volta chiamata visualizza il contenuto del vettore appartenente al *main*

← A questo punto viene richiamata la funzione *visualizza* e gli viene passata una copia dell'indirizzo del vettore *vet*.

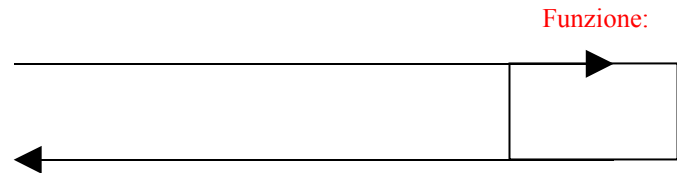
## Funzioni che non restituiscono valori:

In alcuni casi le funzioni (quelle informatiche) non restituiscono nessun valore al chiamante, ma svolgono compiti precisi.



Programma  
*main()* o qualsiasi  
altra funzione

La funzione questa volta non restituisce nessun valore, ma svolge un servizio al chiamante.



La funzione questa volta non restituisce nessun valore, ma svolge un servizio al chiamante.

Vediamo un esempio:

```
#include <stdio.h>
main()
{
    int numpunti;
    void stamppunt(int n); /* Prototipo di funzione */

    printf("\nImmetti il numero di punti da stampare\n");
    scanf("%d", & numpunti);

    stamppunt(numpunti); /* Viene chiamata la funzione */
}

/* FINE MAIN */

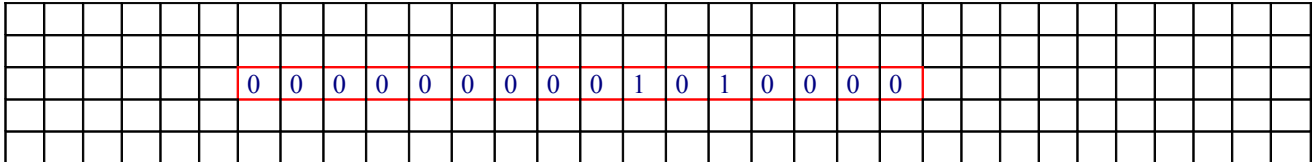
void stamppunt(int n)
{
    while(n-->0){
        printf("\n.\n"); /* La funzione stampa a video una serie di punti */
    }
}
```



`pippo = 'P';`

Assegnando il carattere 'P' alla variabile `pippo` in quello spazio di memoria viene scritto il valore in binario corrispondente a 80 che è associato al carattere 'p' maiuscolo.

RAM



**Tipo di dato                      Numero di Byte occupati in Ram**

char	1
short	2
int	4
long	4
float	4
double	8
long double	12

Questa a fianco mostra i tipi di dato più usati e le dimensioni che occupano nella memoria. Va però detto che lo spazio occupato in RAM varia a seconda del tipo di calcolatore usato e del compilatore.

char }  
short }  
int }      Contengono sempre valori interi con o senza segno  
long }

Char → con 8 bit si possono rappresentare fino 256 valori nel caso specifico caratteri; infatti  $2^8 = 256$

Short → in questo caso si rappresentano valori numeri che vanno da -32768 a 32767; infatti  $2^{16} = 65536$  che corrisponde a  $32768 + 32767 + 1 = 65536$

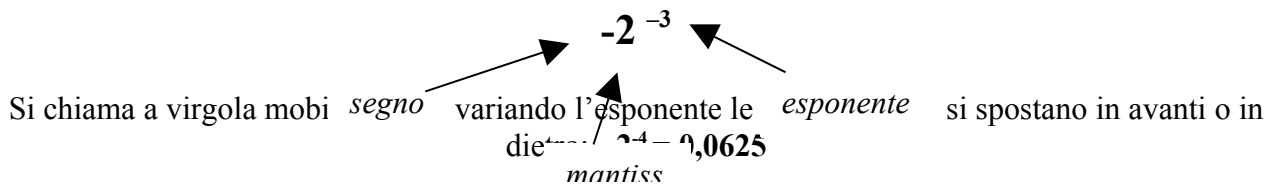
Int → con  $2^{32}$  si possono rappresentare interi da - 2.147.483.648 a 2.147.483.647

## A virgola mobile

Non tutti i problemi sono risolvibili usando numeri interi; spesso occorre rappresentare numeri frazionari, si usano allora i numeri reali. Un numero reale è una grandezza analogica (continua) e quindi è rappresentabile solo in modo approssimato. (Tra due numeri dati, diversi tra loro, è sempre possibile trovarne un terzo che sia maggiore del primo e minore del secondo o viceversa; ciò invece non è sempre possibile nel campo dei numeri interi).

### Esistono due forme per rappresentare un numero reale:

- Segno, parte intera, parte decimale (rappresentazione in virgola fissa) → **-1,03 0,125**
- Segno, mantissa, esponente (rappresentazione **in virgola mobile**) →  $-2^{-3} = 0,125$



In C i numeri con la virgola vengono memorizzati in appositi tipi di variabili che usano la notazione *in virgola mobile*:

float  
duble  
long duble } Contengono valori con la virgola e il segno

(senza entrare nei particolari è fondamentalmente simile all'esempio visto sopra:  $-2^{-2}$ ). Tali valori vengono però accettati da tastiera e visualizzati a schermo con la rappresentazione *Segno, parte intera, parte decimale*; esempio: -0,125.

Lo spazio in byte che generalmente queste variabili occupano in RAM (dipende dal compilatore e dalla macchina) è il seguente:

float 4  
double 8  
long double 12

I numeri reali, in C, si indicano separando la parte intera da quella frazionaria con un punto decimale, secondo la notazione anglosassone:

3.03232 è un reale

Vediamo in questo esempio come si dichiara e si visualizza una variabile a *virgola mobile* di tipo *double*:

```
#include <stdio.h>
#include <math.h>

main()
{
    double valore,radice;

    printf ("\nimmetti il valore\n");
    scanf ("%lf", & valore);

    radice=sqrt(valore);

    printf("\n%lf", radice);
}
```

In quest'altro esempio vediamo invece variabili a *virgola mobile* di tipo *float*:

```
void calclaLeX(float discrim, float b, float a)
{
    double rad,
    pas=(double)discrim;

    printf("\n%lf", pas);

    rad=sqrt(pas);

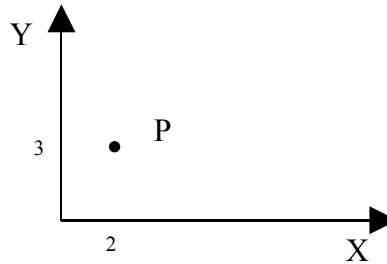
    printf("\n%lf", rad);
}
```

**Strutture:**

In molte situazioni, una variabile non è sufficiente a per descrivere un oggetto; ad esempio una posizione sul piano cartesiano è rappresentata da due coordinate. La si può quindi identificare con due variabili:

la coordinata orizzontale X  
la coordinata verticale Y

Punto P = (2, 3)



Ecco come potrebbe essere definita una la struttura che descrive una posizione nel piano cartesiano:

```
struct posizionePianoCatesiano{ /*Dichiarazione di templete */
    float x;
    float y;
};
```

Successivamente questo nostro tipo di variabile definito da noi, possiamo usarlo come qualsiasi altro tipo di varabile predefinito dal C (vedi *int*, *char*, *double*, ecc.):

```
struct posiz p;
struct posiz q;
```

Vediamo un esempio concreto di creazione e utilizzo di struttura :

```
#include <stdio.h>
main()
{
    struct equazione{ /*Dichiarazione di templete */
        float x;
        float c;
    };

    struct equazione eq; /* Dichiaro una variabile di tipo equazione */
                        /* 'equazione' Š il nome che ho dato io, potevo */
                        /* anche chiamarla 'pippo', in qusto caso dichiravo */
                        /* una variabile di tipo pippo */

    double epselon;

    printf("\nQuesto programma calcola la y di una equazione\n");
    printf("di primo grado nella forma y=x+c)\n");
    printf("\nImmetti il valore della x, e poi il valore della c\n");
    scanf("%f", & eq.x);
    scanf("%f", & eq.c);

    epselon=eq.x+eq.c; /* Mi calcolo la y */

    printf("\nLa y dell' equazione inserita Š: %f", epselon);
}
```

Vediamo un altro esempio di utilizzo di struttura :

```

#include <stdio.h>
//..... Template delle strutture .....
struct equazione{ // Dichiaro dei tipi di variabile che sono
    float a; // utilizzabili da qualsiasi funzione
    float b;
    float c;
};

struct zeriDellEquazione{ /* Sono i valori di x che rendono y = 0 */
    float x1;
    float x2;
};
//.....

float calcolaDiscriminante(struct equazione e){
    float discriminante;
    discriminante = (e.b*e.b)-4.0*(e.a*e.c);
    printf("\n%f", discriminante);
    return discriminante;
}

main()
{
    struct equazione eq;

    struct zeriDellEquazione soluzioni;

    float discrim;

    printf("\n\n\n.....\n");
    printf("\nQuesto programma verifica se una equazione del tipo:");

    printf("\n");
    printf("
    _____");
    printf("\n
    |                                     |");
    printf("\n
    |          aX^2 + bX + c = 0          |");
    printf("\n
    |_____");

    printf("\nha soluzioni reali o meno.");
    printf("\n\n");

    printf("\nImmettere rispettivamente i valori a, b, c\n");
    scanf("%f %f %f", & eq.a, & eq.b, & eq.c);

    discrim=calcolaDiscriminante(eq);

    if (discrim<0.0){
        printf("\nSoluzioni non reali\n");
    }
    if (discrim==0.0){
        printf("\nSoluzioni reali e coincidenti\n");
    }
    if (discrim>1.0){
        printf("\nSoluzioni reali e distinte\n");
    }
}

```

**Note legali:**

1. Queste dispense sono liberamente utilizzabili e pubblicabili sul web, a patto che:
  - a) Non vengano rimosse queste note legali.
  - b) Non venga rimosso il link al sito web della Art Net presente alla fine del documento.
  - c) Non venga rimossa la scritta “www.art-net.info” .
2. Non è permessa la pubblicazione su carta (riviste, libri, stampa, ecc.), salvo esplicita autorizzazione.
3. La Art Net (studio di consulenza informatica che si occupa di: sviluppo software conto terzi o per clienti finali, di outsourcing e body rental, di docenze e corsi d’informatica, di creazione siti web, di sviluppo applicativi gestionali tradizionali o web based) detiene tutti i diritti di copyright.
4. Nel caso di uso di questo materiale, non conforme a queste note, la Art Net perseguirà civilmente e penalmente i trasgressori.
5. Sebbene questi appunti siano stati redatti con coscienza, cura e buona fede, la Art Net non può assumersi alcuna responsabilità circa l’attendibilità del loro contenuto.

Art Net di Arrigoni Roberto via Giovanni XXIII, 4 Civita Castellana (Viterbo) P.I. 01598360566  
fax 0761 51.51.11