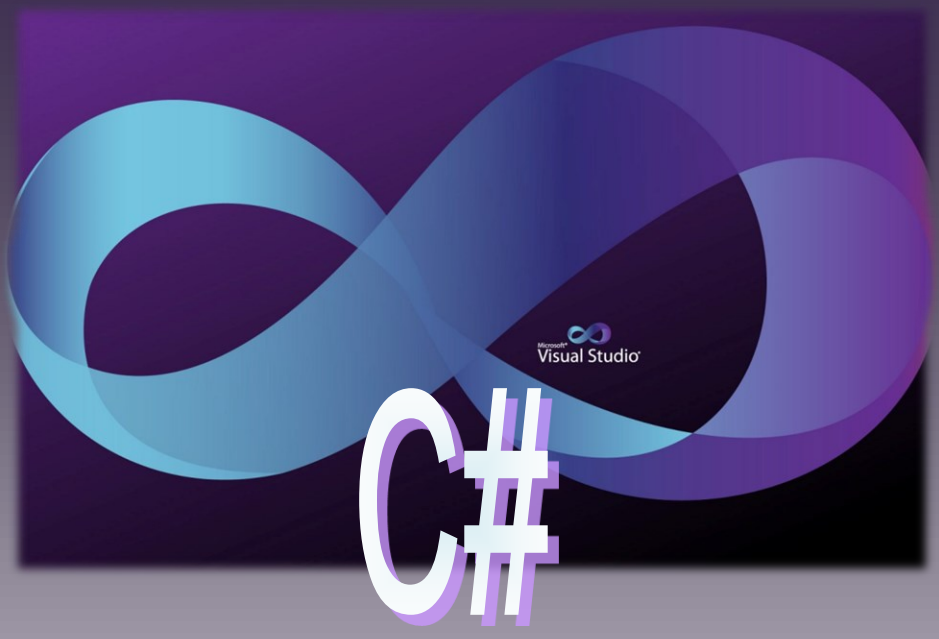
ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI



CORSO DI PROGRAMMAZIONE

INTRODUZIONE AI METODI

DISPENSA 07.01

07-01_Metodi_[ver_15]

Versione del: 04/05/2023
Revisione numero: 15

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

INTRODUZIONE AI METODI

SCOPO E FUNZIONAMENTO DEI METODI

FUNZIONI E METODI

PREMESSE SUL NOME METODO INVECE DI FUNZIONE

Programmazione ad Oggetti

In **C#** quasi tutto è un **oggetto**. Quindi quando si discute di un dato, di un procedimento, di un elemento di programmazione imperativa e procedurale, ci stiamo riferendo in realtà ad un "pezzo" di oggetto.

Per questo è improprio discutere di funzioni in C# perché in realtà sono elementi di un oggetto.

A differenza del C++ quindi parleremo di metodi, invece che di funzioni.

D'altra parte in tutto questo periodo e in questa dispensa useremo i termini "funzione" e "metodo" come sinonimi.

Quando si studia **una funzione** occorre distinguere con chiarezza tra diverse prospettive dell'analisi. Una prima grande distinzione è quella tra definizione e invocazione.

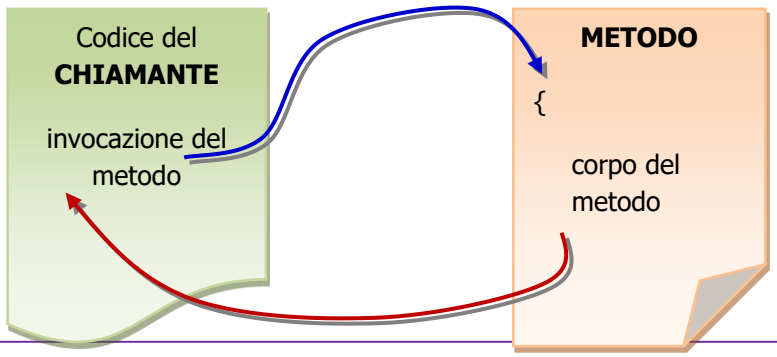
METAFORA DELLA FUNZIONE COME UN LAVORATORE

La funzione va pensata come un soggetto che ha l'incarico di svolgere un compito e deve restituire un risultato a qualcuno.

Proviamo a pensare alla seguente situazione: in una cittadina le persone che vogliono ristrutturare un edificio devono presentare il progetto al comune e il comune risponde con un giudizio SI oppure NO. Il comune incarica un suo impiegato di svolgere questo compito: quando gli viene passato un progetto, egli lo valuta e restituisce al richiedente un documento in cui si autorizza (SI) o meno (NO) l'esecuzione del progetto edilizio.

In questa metafora:

- l'impiegato del comune è **la funzione**
- il cittadino che presenta richiesta edilizia è detto **chiamante**
- l'impiegato per poter svolgere il suo compito deve ricevere un progetto da esaminare; il progetto rappresenta **un parametro**! senza parametro non può lavorare
- l'autorizzazione rappresenta **il risultato** atteso: in pratica ciò che la funzione restituisce al chiamante
- quando un cittadino presenta la domanda all'impiegato sta **chiamando** (invocazione) **la funzione**
- quando il comune spiega al suo impiegato cosa deve fare, ovvero lo istruisce sul compito da svolgere, è il momento di **definizione della funzione**



Il chiamante invoca la funzione, tramite il suo nome; la funzione svolge un lavoro e poi termina (con un risultato o meno, dipende dalla funzione); il controllo ritorna al chiamante che sfrutta l'esito della funzione e, se c'è, può usare il risultato della funzione.

PERCHÉ USARE LE FUNZIONI?

Usare le funzioni offre diversi vantaggi che possiamo riassumere nei seguenti:

1) **Assegnare un nome** a un compito specifico: senza funzioni avremmo gruppi di istruzioni da analizzare ogni volta per sapere quale sia il loro compito/scopo; con le funzioni posso raggruppare alcune istruzioni e attribuire al gruppo un nome, che possibilmente esprima il loro scopo. Per esempio: ColoraFinestra potrebbe essere un nome di funzione che, guarda caso, colora una finestra.
2) **Richiamare la funzione più volte**: se lo stesso compito (per esempio ordinare un vettore di numeri) deve essere svolto più volte (magari su vettori diversi) conviene definire una funzione con un certo nome e poi chiamarla più volte da diversi punti. Per esempio: se ho molti vettori posso chiamare la stessa funzione OrdinaVettore su un vettore diverso ogni volta.
3) **Pulizia del codice**: se un programma è abbastanza grosso, conviene suddividere il programma in frammenti, ciascuno individuato da una funzione. Spesso, addirittura, una funzione importante può richiamare altre funzioni minori per svolgere piccoli compiti. Per esempio: un videogioco può essere gestito da una funzione principale che richiama le funzioni che gestiscono un personaggio principale, gli avversari o l'ambiente di gioco; la funzione relativa al personaggio principale può, a sua volta, richiamare una funzione per il movimento, una per le azioni, una per gestire punteggi o livelli, ecc.

DEFINIZIONE

È il momento in cui il programmatore stabilisce il comportamento della funzione. Di solito questo momento viene accorpato a quello della dichiarazione, che corrisponde al momento in cui stabilisco un nome per la funzione, stabilisco il tipo della funzione e decido il numero ed il tipo dei parametri della funzione.

SINTASSI GENERALE DI UNA FUNZIONE

La sintassi generale di un metodo in C# è la seguente:

```
//dichiarazione di una funzione in C#                    Firma della funzione

<descrittore> <tipo> NomeMetodo (elenco_parametri_OPZIONALE)
{
      //corpo della funzione                             Corpo della funzione
      <corpo>
}
```

Descrittore

È un modo che indica come lavora la funzione. A questo livello esamineremo solo due possibili descrittori, che si possono usare insieme, oppure usare separatamente oppure non usare per niente.

Un descrittore è **static**. Non ci occuperemo in questo momento del significato di static, ma osserveremo che è necessario usarlo per i metodi definiti in modalità Console, mentre lo ometteremo in Visuale.

L'altro descrittore è **public**. Non ci occuperemo in questo momento del significato di public, ma osserveremo che è necessario usarlo SEMPRE.

Esempio

```
//dichiaro una funzione in C#
static public int Rende18() //dichiarazione di intestazione
{                       //corpo della funzione
     return 18; //rende 18
}
```

Tipo

Determina se la funzione rende un dato oppure no. Se la funzione non rende alcun dato, allora devo indicare **void** nel tipo. Se la funzione rende un dato allora occorre indicare il tipo del dato atteso. Per esempio: una funzione che rende il massimo tra due interi deve essere di tipo intero; una funzione che prende due variabili e scambia il contenuto non rende alcun dato ed è di tipo void.

In Visual C# una funzione può rendere dati di tipo valore (int, double, bool, ecc.) stringhe e dati di tipo riferimento come vettori, matrici, oggetti.

Esempio

```
// funzione in C# che rende il massimo tra due interi
static public int Massimo(int x , int y) //dichiarazione di intestazione
{                  //corpo della funzione
     if (x > y)
           return x; //rende il valore di x
     else  return y; //rende il valore di y
}
//dichiaro una funzione in C#
static public void Scambia(ref int x, ref int y) // intestazione
{                  //corpo della funzione
     int t = x;
     x= y;
     y = t;
}
```

Una funzione in C# può restituire sia tipi valore (come int, bool, double, strutture) oppure string (string) oppure riferimento (int[], oggetti).

Nome

Determina come si chiama la funzione. È indispensabile dichiarare il nome della funzione e serve anche per invocarla in seguito.

Lista parametri

Una funzione può avere nessun parametro (ovvero niente lista parametri), oppure un parametro oppure molti parametri. I parametri servono per scambiare dati con l'esterno. Se ci sono molti parametri, vanno separati da virgole. Sui parametri ritorneremo in seguito in questa dispensa, spiegando modi e tipi di passaggio.

Esempio 1.

VC#

```
//funzione senza parametri
static public int Rende18()
{
      return 18;
}
```

Esempio 2.

VC#

```
// funzione con 1 parametro
static public int Quadrato(int x)
{
      return x*x;
}
```

Esempio 3.

VC#

```
// funzione con 2 parametri
static public int Massimo(int x , int y)
{
      if (x > y)
            return x;
      else  return y;
}
```

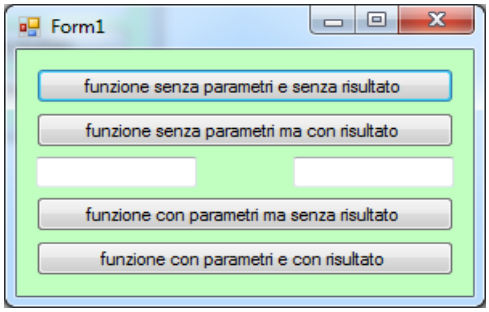
Esempio 4.

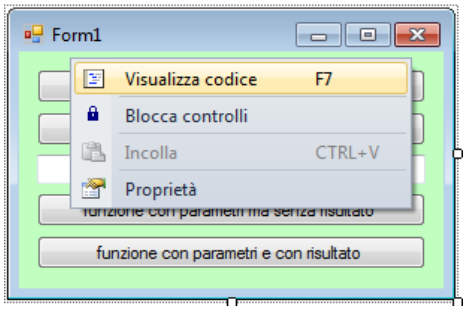
VC#

```
// funzione con 2 parametri
static public void Scambia(ref int x, ref int y)
{
      int t = x;
      x= y;
      y = t;
}
```

ESERCITAZIONE GUIDATA PARTE PRIMA

In questo esercizio proveremo a dichiarare alcune funzioni.





- Prepara un nuovo progetto con una form1 simile alla figura
- Poi apri il codice della applicazione (clic destro sulla form1 e scegli Visualizza codice, oppure F7)
- Posiziona il cursore di scrittura prima del costruttore Form1 e scrivi:

```
public partial class Form1 : Form
{
    //dichiarazioni dei metodi
    public void ColoraFinestra()
    {
        this.BackColor = Color.Red;
    }

    public Form1()
    {
        InitializeComponent();
    }
}
```

Dichiarazione di metodo
Intestazione (firma)
Corpo del metodo

- Continua a dichiarare un secondo metodo; sotto il primo, scrivi:

```
public void MostraNelTitolo (string frase)
{
    this.Text = frase;
}
```

- Continua a dichiarare un terzo metodo; sotto il precedente, scrivi:

```
public string RestituisciNome()
{
    Random estrai = new Random();
    if (estrai.Next(1, 3) == 1)
        return "Ares";
    if (estrai.Next(1, 3) == 2)
        return "Ephest";
    if (estrai.Next(1, 3) == 3)
        return "Aphrodith";
    return "Impossibile";
}
```

- Continua a dichiarare un quarto metodo; sotto il precedente, scrivi:

```
public double CalcolaMedia (int numero, int altroNumero)
{
      double media;
      media = (numero + altroNumero) / 2.0;
      return media;
}
```

- Con questo esercizio hai dichiarato 4 diversi metodi,ma ancora il programma non li usa e non fa nulla!

INVOCAZIONE

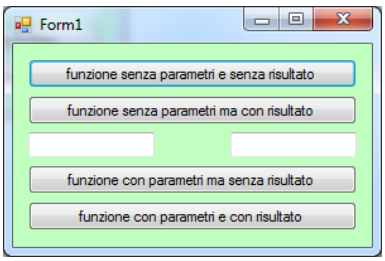
È il momento in cui il programmatore stabilisce di utilizzare la funzione. Questo momento comporta una deviazione del flusso di elaborazione e costringe il programma ad eseguire il corpo della funzione.

In questa fase accadono i seguenti fatti:

1. il codice del chiamante viene temporaneamente interrotto
2. gli argomenti indicati nella chiamata vengono passati ai parametri definiti con la funzione
3. viene eseguito il corpo della funzione
4. quando il codice del corpo termina OPPURE si incontra un comando **return** il controllo ritorna al chiamante
5. il codice del chiamante riceve il risultato della funzione e può proseguire

ESERCITAZIONE GUIDATA PARTE SECONDA

In questo esercizio proveremo a invocare le funzioni dichiarate in precedenza. Per invocare una funzione è necessario utilizzare il suo nome (identificatore) le parentesi tonde e, al loro interno, indicare gli argomenti compatibili in numero e tipo.



- Riapri il precedente progetto e seleziona il primo pulsante
- Associa al primo pulsante il codice seguente:

```
ColoraFinestra();//invocazione della funzione
```

- Associa al secondo pulsante il codice seguente:

```
string s;
s = RestituisciNome(); //invocazione della
funzione
MessageBox.Show (s);
```

- Associa al terzo pulsante il codice seguente:

```
string s;
s = "Ottimo Motto ";
MostraNelTitolo (s);
```

- Associa al quarto pulsante il codice seguente:

```
int x = 13;
int y = 17;
double media;
media = CalcolaMedia (x, y)
MessageBox.Show ("risultato: " + media);
```

- Prova il progetto.

FUNZIONI SENZA PARAMETRI

Abbiamo visto che una funzione va analizzata sotto due diversi punti di vista: dichiarazione ed invocazione. La dichiarazione è il momento in cui il programmatore stabilisce il compito che la funzione deve svolgere. In questa fase il programmatore deve decidere il nome della funzione che sarà indispensabile per invocarla in seguito.

FUNZIONI SENZA RISULTATO NÉ PARAMETRI

Una funzione senza risultato non rende niente; il fatto di non rendere niente, non equivale a non fare niente. Per comprendere meglio ipotizziamo un'azienda che assuma dei ragazzi per distribuire volantini per la città: i ragazzi prendono i volantini e vanno in giro a consegnarli, ma quando ritornano dal datore di lavoro non riportano nulla; in pratica hanno svolto un lavoro, ma non hanno nulla da consegnare a chi li ha chiamati.

Una funzione senza risultato si dice di tipo **void**. Il valore **void** (letteralmente vuoto) equivale a nessun valore.

Una funzione senza parametri è una funzione che lavora senza informazioni dal chiamante; quando una funzione è senza parametri significa che dopo il nome, tra le parentesi tonde non c'è nulla (ma le parentesi sono sempre obbligatorie).

DICHIARAZIONE ED INVOCAZIONE

Vediamo il caso di alcune funzioni senza risultato né parametri:

DICHIARAZIONE

```
//funzione senza risultato né parametri
public void ColoraFinestra()
{
  if (this.BackColor == Color.Red)
     this.BackColor = Color.Blue;
  else
     this.BackColor = Color.Red;
}
```

INVOCAZIONE

```
//funzione senza risultato né parametri
//puoi invocarla da un pulsante button1
ColoraFinestra();
```

(1) Nella dichiarazione compare la parola **void**, ovvero senza risultato; significa che la funzione non rende valori e che può essere invocata come se fosse un comando.

(2) Il nome della funzione è obbligatorio, serve per individuare la funzione senza ambiguità, e serve per poterla invocare senza problemi. Quando si invoca una funzione si usa il suo nome.

(3) Tra le parentesi non c'è nulla, ovvero senza parametri. Le parentesi sono obbligatorie sia in fase dichiarativa che invocativa; se una funzione è dichiarata senza parametri (con parentesi vuote) DEVE essere invocata senza parametri (con parentesi vuote).

FUNZIONI CON RISULTATO E SENZA PARAMETRI

Una funzione con risultato deve rendere un valore; il valore sarà restituito al codice che effettua l'invocazione, detto chiamante.

Per comprendere meglio ipotizziamo che un docente chieda ad un alunno di girare per la scuola per fare una colletta per una buona opera; l'alunno va in giro, raccoglie i soldi ma infine deve tornare dal docente per consegnarli a lui; in pratica ha svolto un lavoro che rende un frutto, e questo esito va reso a colui che ha richiesto il lavoro. Una funzione con risultato deve specificare il tipo del valore restituito.

DICHIARAZIONE ED INVOCAZIONE

Vediamo il caso di alcune funzioni con risultato ma senza parametri:

DICHIARAZIONE

```
//funzione con risultato senza parametri
public int EstraiNumeroPari()
{          (4)
      Random estrai = new Random();
      return estrai.Next(1,10)*2;
}
```

INVOCAZIONE

```
//funzione con risultato senza parametri
//puoi invocarla da un pulsante button1
int numero;
numero = EstraiNumeroPari();
  (4)
```

(4) Nella dichiarazione, al posto della parola void, c'è un tipo che corrisponde al valore da rendere. Quando si invoca la funzione, usando il suo nome, è possibile usare il valore che renderà, per esempio per assegnarlo ad una variabile. Nell'esempio precedente il chiamante dichiara una variabile (numero) in cui sarà depositato il valore reso dalla funzione, al termine della sua esecuzione.

Una funzione con un tipo, deve avere un **return** dentro il corpo per assicurare che essa renda un valore.

Esempio di funzione di tipo string

DICHIARAZIONE

```
//funzione con risultato senza parametri
public string Battesimo()
{          (5)
      Random estrai = new Random();
      int valore = estrai.Next(3);
      if (valore == 0)
            return "Artemisia";
      if (valore == 1)
            return "Brunilde";
      if (valore == 2)
            return "Cleopatra";
      return "Impossibile";
}
```

INVOCAZIONE

```
//funzione con risultato senza parametri
//puoi invocarla da un pulsante button1
string nome;
nome = Battesimo();
  (5)
```

(5) Nella dichiarazione, il tipo restituito è string; nella invocazione il valore restituito viene copiato nella variabile nome; si può notare che dopo la parola return ci deve essere una stringa.

Esempio di funzione di tipo array

DICHIARAZIONE	INVOCAZIONE

`//funzione con risultato`
`//senza parametri`
`public int[] CreaVettore()`
`{`
`  int[] vettore = new int[100];`
`  Random estrai = new Random();`
`  for (int i=0; i<100; i++)`
`   vettore [i] = estrai.Next(10,100);`
`  return vettore;`
`}`
	`//funzione con risultato`
	`//senza parametri`
	`//puoi invocarla`
	`//da un pulsante button1`
	`int[] risultato;`
	`risultato = CreaVettore();`

(6) Nella dichiarazione, il tipo restituito è un vettore di interi; anche in questo caso, si può osservare che il tipo restituito deve coincidere col tipo che segue la parola chiave return.

FUNZIONI CON PARAMETRI

In alcuni casi è necessario utilizzare dei parametri. I parametri sono un modo con cui è possibile far comunicare la funzione (il metodo) con il chiamante (il codice che invoca la funzione). I parametri sono delle locazioni (come delle variabili), che esistono solo all'interno della funzione, in cui ospitare i dati che saranno oggetto della comunicazione tra funzione e chiamante. I parametri vanno dichiarati in fase dichiarativa. Quando si invoca la funzione è necessario utilizzare tanti argomenti quanti sono i parametri dichiarati, e gli argomenti devono essere dello stesso tipo.

FUNZIONI SENZA RISULTATO COI PARAMETRI

È il caso di ricordare che una funzione senza risultato non rende niente (come già spiegato nel precedente capitolo). Una funzione senza risultato è di tipo **void**.

Una funzione con i parametri lavora con informazioni passate dal chiamante; quando una funzione è con parametri significa che dopo il nome, tra le parentesi tonde sono elencati i parametri. I parametri nel passaggio per valore (il passaggio più elementare) sono elencati separati da virgole, ciascuno nella forma <tipo nome>: in pratica è come dichiarare della variabili (tra le parentesi) senza assegnare loro alcun valore iniziale.

DICHIARAZIONE ED INVOCAZIONE

Vediamo il caso di alcune funzioni senza risultato con dei parametri..

Esempio 5.

DICHIARAZIONE

```
//funzione senza risultato né parametri
public void ColoraFinestra (int parametro)
{        (1)       (2)              (3)
     if (parametro > 0)
           this.BackColor = Color.Blue;
     else
           this.BackColor = Color.Red;
}
```

INVOCAZIONE

```
//funzione senza risultato né parametri
//puoi invocarla da un pulsante button1
ColoraFinestra(13);
     (2)
int argomento =  -17;
ColoraFinestra(argomento);
                  (3)
```

(1) Nella dichiarazione compare la parola **void**, ovvero senza risultato; significa che la funzione non rende valori e che può essere invocata come se fosse un comando.

(2) Il nome della funzione è obbligatorio, serve per individuare la funzione senza ambiguità, e serve per poterla invocare senza problemi. Quando si invoca una funzione si usa il suo nome.

(3) Tra le parentesi compare il parametro; il parametro somiglia a una variabile; al momento della invocazione in chiamante specifica il valore che deve copiare nel parametro; generalmente il valore specificato dal chiamante prende il nome di argomento.

Nota: se il parametro è intero, allora l'argomento deve essere intero!

Esempio 6.

DICHIARAZIONE

```
//funzione senza risultato con parametri
public void Colora (int x, int y)
{       (1)   (2)        (3)
     if (x > y)
          this.BackColor = Color.Blue;
     else
          this.BackColor = Color.Red;
}
```

INVOCAZIONE

```
//funzione senza risultato con parametri
//puoi invocarla da un pulsante button1
Colora (13 , 17);
  (2)      (3)
int a =  -17;
Colora (a , -11);
  (2)      (3)
```

(1) Nella dichiarazione compare la parola **void**, perciò la funzione è senza risultato.

(2) Il nome della funzione dovrebbe esemplificare il suo scopo.

(3) Nella definizione, tra le parentesi compaiono più parametri, ciascuno col suo tipo; nella invocazione ci sono argomenti in numero e tipo coerenti con la definizione.

Esempio 7.

DICHIARAZIONE

```
//funzione senza risultato con parametri
public void MutaBottone (string s, bool b)
{                                    (4)
      Button1.Text = s;
      Button1.Enabled = b;
}
```

INVOCAZIONE

```
//funzione senza risultato con parametri
//puoi invocarla da un pulsante button1
bool abilita = true;
MutaBottone ("ciao" , abilita);
                      (4)
MutaBottone ("salve" , !abilita);
                      (4)
```

(4) Nella definizione, tra le parentesi compaiono più parametri, ciascuno col suo tipo; nella invocazione ci sono argomenti in numero e tipo coerenti con la definizione.

FUNZIONI CON RISULTATO E COI PARAMETRI

Come già detto, una funzione con risultato rende qualcosa (come già spiegato nel precedente capitolo).

Quando una è definita funzione con risultato e con parametri, significa che la funzione deve avere dati dal chiamante per poter lavorare, e alla fine della sua elaborazione restituisce al chiamante un valore del tipo specificato.

DICHIARAZIONE ED INVOCAZIONE

Vediamo il caso di alcune funzioni con risultato e con dei parametri..

Esempio 8.

DICHIARAZIONE

```
//funzione senza risultato né parametri
public double Media (int a, int b)
{          (1)    (2)          (3)
      return ((a + b)/2.0);
}
```

INVOCAZIONE

```
//funzione senza risultato né parametri
double m =  0.0;
m = Media (13 , 17);
```
(1) (2) (3)

(1) Nella dichiarazione compare la parola **double**, cioè restituisce un numero con la virgola; significa che la funzione dovrebbe (ma non è obbligatorio) essere invocata per utilizzare il risultato, per esempio in una assegnazione o in una espressione.

(2) Il nome della funzione è obbligatorio.

(3) Tra le parentesi sono elencati i parametri; quando si scrive un'invocazione se devono elencare gli argomenti nello stesso numero dei parametri e devono avere tipo compatibile.

I PARAMETRI

La trattazione dei parametri deve essere affrontata con particolare cura. In effetti occorre spiegare, con dovizia di particolari ed esempi, il significato dei parametri e le diverse modalità di passaggio.

Anzitutto occorre ricordare che i parametri costituiscono il principale strumento di comunicazione fra un metodo (una funzione) e il chiamante (il programma che invoca tale funzione). questa comunicazione può avvenire in diversi modi, con differenti significati. In Visual C# esistono tre diversi modi di passare i parametri:

- Passaggio per valore
- Passaggio per riferimento
- Passaggio per risultato

Ciascuno di questi modi di passaggio deve essere usato quando è opportuno e facendo attenzione alle sottili differenze che li contraddistinguono.

In un metodo è possibile avere più parametri e ciascuno di questi deve indicare il suo specifico modo di passaggio.

La lista dei parametri quindi appare con la seguente sintassi:

Lista parametri = <modo><tipo><nome> , <modo><tipo><nome> , . . . , <modo><tipo><nome>

PASSAGGIO PER VALORE

Sintassi

Il passaggio standard è quello per valore; se non si indica alcun modo di passaggio, il compilatore lo considera un passaggio per valore. Nel passaggio per valore il parametro deve indicare sempre il tipo seguito dal nome. Il passaggio per valore si scrive nel seguente modo:

parametro = <**tipo**><**nome**>

Esempio

Consideriamo una funzione che debba restituire il valore più piccolo tra due valori interi ricevuti in ingresso; allora la funzione si può schematizzare come segue:

allora la funzione assume il seguente aspetto:

```
public int Minimo (int x , int y)
{       (1)   (2)          (3)
     if (x < y)
           return x;
     else
           return y;
}
```

(1) Il tipo restituito è un intero: significa che al termine dell'elaborazione sarà reso un valore intero.

(2) Il nome è obbligatorio e dovrebbe esprimere l'obiettivo che si propone.

(3) I parametri sono entrambi passati per valore; ciascuno deve indicare il tipo e il nome.

Di seguito sono proposte alcuni diversi modi di invocazione del metodo Minimo():

Invocazione 1

```
//esempio invocazione 1
int min;
min = Minimo ( 13, -17);
Text = "" + min ;
```

Invocazione 2

```
//esempio invocazione 2
int a = -13;
int b = 7;
int ris = Minimo ( a, b);
Text = "" + ris ;
```

Invocazione 3

```
//esempio invocazione 3
int z;
z = Minimo (Minimo (1,2), 3);
Text = "" + z ;
```

Significato e funzionamento del passaggio per valore

Il passaggio per valore comporta la copia del valore degli argomenti nei parametri. In altre parole, quando si invoca un metodo con parametri passati per valore, si devono valutare gli argomenti e stabilire il loro valore; questo valore viene **copiato** negli identificatori dei parametri, che tuttavia restano distinti dagli argomenti.

Nell'ultimo esempio proposto, nel caso della prima invocazione, il metodo Minimo è invocato con due argomenti, i valori 13 e -17; questi valori sono copiati nei parametri **x** ed **y**; quindi prima di iniziare l'elaborazione del metodo **x** vale 13 e **y** vale -17; quando inizia l'elaborazione il metodo controlla se **(x<y)** il cui valore è false; il metodo esegue il ramo else che impone al metodo di terminare col risultato -17: il valore -17 è reso al chiamante che copia -17 nella variabile **min** e usata in seguito.

Nel caso della seconda invocazione, il metodo Minimo è invocato con due argomenti: le variabili **a** e **b**. Quindi si procede prima a valutare la variabile **a** che vale -13 e la variabile **b** che vale 7. Il valore -13 è copiato nel parametro **x**, mentre il valore 7 è copiato nel parametro **y**: si deve notare che adesso gli spazi di memoria di **a** e **b** sono del tutto distinti da quelli di **x** e **y**. Se la funzione modifica **x** oppure **y**, non ci sarà alcun effetto né su **a** né su **b**. Il metodo lavorerà come ci si aspetta, ed infine rende -13.

Il caso della terza invocazione è lasciato per esercizio, ma occorre osservare che in realtà ci sono due distinte invocazioni e che prima di procedere con la seconda è necessario far terminare la prima.

PASSAGGIO PER RIFERIMENTO

Premessa

Prima di analizzare il passaggio per riferimento, consideriamo la seguente funzione, che si propone di scambiare i valori contenuti in due variabili:

Esempio 9.

DICHIARAZIONE

```
public void Scambio (int x , int y)
{
      int tmp = x;
      x = y;
      y = tmp;
}
```

INVOCAZIONE

```
int a = 13;
int b = 17;
Scambio ( a , b );
MessageBox.Show ("a: " + a + " b: " + b);
Scambio ( a , 1 );
MessageBox.Show ("esito: " + a);
```

Proviamo a analizzare cosa accade quando si invoca la funzione con gli argomenti a e b: i loro rispettivi valori sono 13 e 17 che sono copiati nei parametri x e y; quindi x vale 13 e y vale 17. Adesso la funzione scambia i valori contenuti in x e y, infine termina. Ma quando si prova a visualizzare i valori contenuti in a e b c'è l'amara sorpresa che i valori NON sono stati scambiati! E lo stesso accade al successivo tentativo di scambio! Ahh!

Purtroppo è stato usato il passaggio per valore, che mantiene separati gli spazi dei parametri da quelli degli argomenti; quindi quando si modificano i parametri, non si ottiene alcun effetto sugli argomenti.

Sintassi

Il passaggio per riferimento, consente di condividere lo spazio degli argomenti con quello dei parametri. Il passaggio per riferimento deve essere reso esplicito, scrivendo il modo **ref** prima della dichiarazione del parametro rispettivo.

Il parametro passato per riferimento deve indicare sempre il tipo seguito dal nome. Il passaggio per riferimento si scrive nel seguente modo:

parametro = <**ref**><**tipo**><**nome**>

Esempio 10.

Vediamo ancora il metodo dello scambio, ma utilizzando il passaggio per riferimento:

```
public void Scambio (ref int x , ref int y)
{
     int tmp = x;
     x = y;
     y = tmp;
}
```

1. Il tipo restituito è un intero: significa che al termine dell'elaborazione sarà reso un valore intero.
2. Il nome è obbligatorio e dovrebbe esprimere l'obiettivo che si propone.
3. I parametri sono entrambi passati per riferimento; e ciascuno deve indicare il tipo e il nome.

Di seguito sono proposte alcuni diversi modi di invocazione del metodo Scambio():

VC#

```
//esempio invocazione 1
int a = 13;
int b = 17;
Scambio (ref a, ref b);
Text = "" + a;
```

La **prima** invocazione agisce bene: si osservi che è necessario indicare ref anche nell'invocazione!

Errore

```
//esempio invocazione 2
int a = 3;
Scambio (ref a, ref 17);
// si solleva un errore!
Text = " errore!" ;
```

La **seconda** invocazione NON può procedere, poiché si solleva un errore di sintassi: una costante non può essere passata per riferimento, poiché non può essere mai modificata!

Warning

```
//esempio invocazione 3
int a, b;
Scambio (ref a, ref b);
//può sollevarsi un errore!
Text = " errore?" ;
```

La **terza** invocazione potrebbe NON procedere, nel caso in cui l'argomento a non abbia un valore iniziale; in pratica prima di usare la variabile come argomento deve avere un valore già specificato! In alcuni casi (come per le variabili globali) la dichiarazione implica l'inizializzazione con un valore di default, ma in generale (come variabili locali) questo non accade e il compilatore solleva un errore.

Significato e funzionamento

Nel passaggio per riferimento viene condivisa la memoria dell'argomento con quella del parametro. In pratica, se un metodo compie una qualsiasi modifica su un parametro passato per riferimento, in realtà si modifica il valore dell'argomento. Il passaggio per riferimento si usa quindi ogni volta che si desidera che gli argomenti riflettano le modifiche operate sui parametri.

Nel passaggio per riferimento è necessario che:

- L'argomento abbia un valore prima di essere utilizzato in una invocazione;
- L'argomento sia una locazione modificabile e, quindi, non può essere una costante;

PASSAGGIO PER RISULTATO

Premessa

Prima di analizzare il passaggio per risultato, consideriamo la seguente funzione, che si propone di porre in una variabile un nome casuale:

Esempio 11.

DICHIARAZIONE

```
public void NomeCasuale (ref string s)
{          1          2                    3
      Random dado = new Random();
      int valore = dado.Next (1,7);
      switch (valore)
      case 1: s = "Abele"; return;
      case 2: s = "Bruto"; return;
      case 3: s = "Cassio"; return;
      case 4: s = "Dante"; return;
      case 5: s = "Esra"; return;
      default: s = "Falstaff"; return;
}
```

La funzione ha l'intento di restituire nel parametro un dato nuovo, che non è la modifica di un dato preesistente nell'argomento, ma crea un dato originale che viene depositato nella locazione argomento indicata al momento dell'invocazione. Proviamo allora ad analizzare cosa accade quando si invoca la funzione.

INVOCAZIONE CON ERRORE

Errore

```
int nome;
NomeCasuale (ref nome);
MessageBox.Show ("esito: " + nome);
```

Nell'invocazione con l'argomento nome, si solleva un errore, poiché il parametro richiede che vi sia già un valore specificato per l'argomento. Questo modo di invocare la funzione non funziona sebbene auspicabile.

INVOCAZIONE 2

Warning

```
int altroNome;
altroNome = "Giuda";
NomeCasuale (ref altroNome);
MessageBox.Show ("esito: " + altroNome);
```

Nella seconda invocazione si risolve il problema indicando un valore iniziale all'argomento («Giuda»), il quale valore però non serve a nulla, salvo che a evitare l'errore. Questo nome è inutile e può creare dubbi in chi dovesse leggere e modificare il programma.

Per risolvere il problema si può utlizzare il passaggio per risultato, che lavora in modo analogo al passaggio per riferimento ma non ha bisogno di specificare un valore iniziale prima dell'invocazione.

Vediamo come si usa il passaggio per risultato.

Sintassi

Il passaggio per risultato, consente di condividere lo spazio degli argomenti con quello dei parametri. Il passaggio per risultato deve essere reso esplicito, scrivendo il modo **out** prima della dichiarazione del parametro rispettivo. Il parametro passato per risultato deve indicare sempre il tipo seguito dal nome. Il passaggio per risultato si scrive nel seguente modo:

parametro = <**out**><**tipo**><**nome**>

Esempio 12.

Vediamo come scrivere il metodo del nome casuale utilizzando il passaggio per **risultato**:

```
public void NomeCasuale (out string s)
{       1        2        3
     Random dado = new Random();
     int valore = dado.Next (1,7);
     s = "Falstaff";
     switch (valore) {
     case 1: s = "Abele"; return ;
     case 2: s = "Bruto"; return ;
     case 3: s = "Cassio"; return ;
     case 4: s = "Dante"; return ;
     case 5: s = "Esra"; return ;
     }
}
```

Di seguito sono proposte alcuni diversi modi di invocazione del metodo NomeCasuale():

VC#

```
//esempio invocazione 1
string nome;
NomeCasuale (out nome);
MessageBox.Show (nome);
```

La **prima** invocazione agisce bene: si osservi che è necessario indicare **out** anche nell'invocazione!

VC#
```
//esempio invocazione 2
string altroNome = "Giuda";
NomeCasuale (out altroNome);
MessageBox.Show (altroNome);
```

La **seconda** invocazione agisce comunque bene, sebbene il nome "Giuda" sia inutile e verrà sostituito.

Errore
```
//esempio invocazione 3
NomeCasuale (out "Giuda");
```

La **terza** invocazione invece NON può operare, si solleva un errore poiché è indispensabile che l'argomento sia una locazione, per esempio una variabile.

Significato e funzionamento del passaggio per risultato

Il passaggio per risultato è simile a quello per riferimento. A differenza del riferimento nel corpo del metodo è necessario che vi sia una assegnazione al parametro, per evitare il rischio che la funzione si concluda col parametro senza alcun valore. Un'altra differenza è che nell'invocazione non è necessario specificare un valore iniziale: infatti spetta al metodo inizializzare l'argomento, proprio forzando l'assegnazione nel suo corpo.

ISTRUZIONE RETURN

Nella funzione c'è una istruzione speciale detta **return**. L'istruzione **return** indica al programma cosa deve rendere la funzione al chiamante. Di seguito alla parola **return** si indica il valore (espressione, da calcolare) che la funzione deve rendere. La parola **return** interrompe l'elaborazione della funzione e il controllo del flusso al chiamante.

Se una funzione è tipizzata (cioè non è **void**) è obbligatorio che ci sia almeno una istruzione **return** che verrà sicuramente eseguita; il compilatore solleverà errori se manca la parola **return** oppure se essa è condizionata da istruzioni come **if**, **switch** o **cicli** che non diano garanzia di una sua elaborazione.

Esempio 13. mancanza del return in funzione tipizzata

Errore
```
public bool Poldo (int x , int y)
{
     while ( x != y )
          if ( x > y )
               x = x - y ;
          else
               y = y - x ;
}
//solleva un errore, perché manca un return!
```

La funzione appena definita solleva un errore di compilazione. Il compilatore segnala che manca l'istruzione **return** che garantisca il valore risultato della funzione. Il programma non può essere avviato.

Esempio 14.

Errore

```
public bool Pippo (int x)
{
      if ( x > 0 )
            return true;
      if ( x < 0 )
            return false;
}
//solleva un errore, perché non è certa!
```

La funzione appena definita solleva un errore di compilazione. Il compilatore segnala che manca una istruzione **return** che garantisca sicuramente il valore risultato della funzione. Il programma non può essere avviato. il compilatore ritiene che vi siano dei casi in cui la funzione non è in grado di rendere un risultato certo (e a ben guardare ha anche ragione).

Se una funzione è void, non è obbligatorio che vi sia una istruzione return; tuttavia non è nemmeno vietato; l'istruzione return sarà usata senza specificare alcun valore, e servirà esclusivamente per terminare l'elaborazione della funzione.

Esempio 15.

VC#

```
public void Pippo (ref int x, ref int y)
{
      while ( x != y )
            if ( x > y )
                  x = x - y ;
            else
                  y = y - x ;
}
//ok, anche senza return
```

Questa funzione non ha necessità dell'istruzione **return** per lavorare correttamente, poiché non è tipizzata come si può notare dal tipo **void** indicato nella firma.

Esempio 16.

VC#

```
public void Cerca (ref int x , ref int y)
{
      if ( x == y )
            return;
      if ( x > y )
            x = x - y ;
      else
            y = y - x ;
      Cerca (ref x , ref y);
}
//ok, perché return NON indica un valore!
```

Anche questa funzione non ha bisogno dell'istruzione **return**, tuttavia il suo uso non è vietato e la funzione viene compilata correttamente. Il comando **return** chiude subito la funzione senza elaborare alcuna altra istruzione del corpo.

RIEPILOGO

Quando si invoca una funzione occorre tenere ben distinti i parametri formali (quelli indicati nella definizione della funzione) dagli argomenti attuali (quelli specificati al momento della invocazione).

In particolare occorre tenere a mente che i parametri formali ESISTONO SOLO nell'ambiente locale della funzione, e fuori da essa NON ESISTONO.

D'altra parte gli argomenti attuali della chiamata esistono solo se sono visibili nell'ambiente del chiamante, ovvero sono utilizzabili dal frammento di codice che sta invocando la funzione.

Quando la funzione viene chiamata il sistema alloca memoria per poter eseguire la funzione. In questa memoria si sistemano spazi adatti sia ai parametri formali che alle variabili locali alla funzione. I parametri sono locali alla funzione che li usa come dei segnaposto per svolgere ipotetici calcoli.

PASSAGGIO PER VALORE

Appena dopo aver creato questo spazio di memoria, il sistema provvede a copiare i valori degli argomenti attuali nelle locazioni dei parametri formali. Si faccia attenzione: le locazioni sono ben distinte anche se i nomi possono coincidere. Quello che avviene è che le locazioni associate ai parametri formali vengono riempite da copie dei valori contenuti negli argomenti attuali.

Quando si esegue l'invocazione i valori degli argomenti vengono copiati nei parametri. Una volta acquisiti i valori la funzione può svolgere i calcoli e, eventualmente, restituire un risultato.

Nel passaggio di parametri per valore non si specifica alcun modo nel parametro. I valori degli argomenti sono copiati nei parametri. I parametri lavorano su copie dei valori, non sugli originali. Per questo eventuali modifiche sui parametri NON HANNO EFFETTO sugli argomenti. Questo effetto può garantire sicurezza in alcuni casi ma non sempre è l'effetto desiderato.

Quando si studia una funzione occorre tenere ben presente quello che "vorrei che facesse" da quello "che effettivamente fa". Se il mio desiderio è di avere una funzione che scambi i valori contenuti in due variabili ricevute come parametri, devo verificare che la funzione realizzata corrisponde a quello che effettivamente fa. Se avessi usato un passaggio per valore, probabilmente non funzionerà.

PASSAGGIO PER RIFERIMENTO

Quando si desidera che gli argomenti passati possano essere modificati dalla funzione, occorre passarli per riferimento. Il passaggio per riferimento permette alla funzione di agire sugli argomenti che il programma chiamante ha passato. In C# il passaggio per riferimento si dichiara utilizzando la parola chiave **`ref`**.

Nell'esempio dello scambio gli argomenti devono essere passati per riferimento; quando viene invocata la funzione in realtà NON c'è alcuna copia di valori, ma si condividono le stesse locazioni di memoria. In pratica ogni volta che la funzione lavora sui parametri, in effetti il lavoro viene compiuto sugli argomenti.

Nel passaggio di parametri per riferimento si specifica il modo ref anche per l'argomento.

Poiché il passaggio per riferimento implica che la funzione possa modificare l'argomento, ne consegue che l'argomento non può essere una costante e deve essere inizializzato con un valore prima di usarlo nell'invocazione.

PASSAGGIO PER RISULTATO

Quando si desidera che gli argomenti possano non essere inizializzati e che spetti alla funzione attribuire loro un valore, allora occorre passarli per risultato. Il passaggio per risultato permette alla funzione di agire sugli argomenti che il programma chiamante ha passato. In C# il passaggio per riferimento si dichiara utilizzando la parola chiave **`out`**.

Poiché il passaggio per risultato suggerisce che la funzione si occupa di preparare il parametro, non occorre prepararlo prima dell'invocazione, neppure di allocare la memoria per variabili di tipo riferimento.

Nel passaggio di parametri per risultato si specifica il modo out anche per l'argomento in occasione dell'invocazione. Le locazioni del parametro condividono quelle degli argomenti. I parametri lavorano proprio sugli argomenti, non su copie.

ESERCIZI

ESERCIZI SULLE FUNZIONI

ESERCIZI SENZA PARAMETRI

Predisporre una form1 con tre pulsanti e tre caselle di testo; si noti che uno stesso pulsante può invocare più di una funzione; poi si risolvano i seguenti esercizi:

1) Dichiarare una funzione che cambi il titolo della finestra aggiungendo una lettera "a" minuscola alla fine;
il button1 la invoca una volta, ma il button2 la invoca due volte
2) Dichiarare una funzione che restituisca un colore casuale scelto tra tre possibili (per esempio Rosso, Verde e Blu); il button1 la invoca una volta
3) Dichiarare una funzione che scambia il testo della textBox1 con la textBox2; il button2 la invoca una volta
4) Dichiarare una funzione che sposta casualmente di 10 pixel in direzione orizzontale la form1; il button3 la invoca una volta
5) Dichiarare una funzione che sposta casualmente di 10 pixel in direzione verticale la form1; il button3 la invoca una volta
6) Dichiarare una funzione decide casualmente se invocare la funzione di spostamento orizzontale oppure quella di spostamento verticale; il button2 la invoca una volta
7) Dichiarare una funzione che restituisca un valore con la virgola compreso tra 0 e 10 (per esempio 3,962 oppure 7,159 oppure 9.001); il button1 la invoca una volta

ESERCIZI CON PARAMETRI

Predisporre una form1 con tre pulsanti e tre caselle di testo; si noti che uno stesso pulsante può invocare più di una funzione; poi si risolvano i seguenti esercizi:

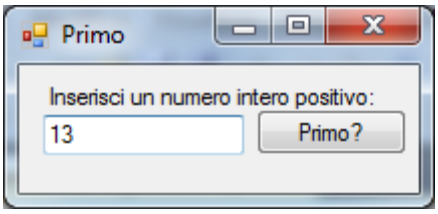
1) Dichiarare una funzione che rende true se due numeri interi sono uguali, altrimenti rende false;
il button1 la invoca e colora di Rosso la finestra se la funzione rende true, altrimenti di Verde
2) Dichiarare una funzione che riceve tre numeri interi e incrementa il minore di +1
il button1 la invoca e mostra nelle caselle di testo l'effetto sui parametri
3) Dichiarare una funzione che rende la concatenazione di due stringhe passate per parametro;
il button2 la invoca una volta e mostra nella terza casella la concatenazione del testo delle prime due
4) Dichiarare una funzione che riceve tre numeri interi e casualmente ne raddoppia uno
il button3 la invoca e mostra nelle caselle di testo l'effetto sui parametri
5) Dichiarare una funzione con quattro parametri; la funzione pone nel terzo parametro il minimo valore tra i primi due, e nel quarto parametro il massimo valore tra i primi due
6) Dichiarare una funzione che riceve tre numeri interi e eleva al quadrato il minore
il button3 la invoca e mostra nelle caselle di testo l'effetto sui parametri
7) Dichiarare una funzione che riceve tre numeri interi e ne rende il loro prodotto
il button3 la invoca e mostra nelle caselle di testo l'effetto
8) Dichiarare una funzione che dato il raggio rende il diametro di una sfera; il button3 la invoca usando le caselle di testo per acquisire il raggio e mostrare il diametro
9) Dichiarare una funzione che dato il raggio rende il volume di una sfera; il button3 la invoca usando le caselle di testo per acquisire il raggio e mostrare il volume
10) Dichiarare una funzione che dato il raggio rende la superficie di una sfera; il button3 la invoca usando le caselle di testo per acquisire il raggio e mostrare la superficie

ESERCIZI SU FUNZIONI SENZA VETTORI

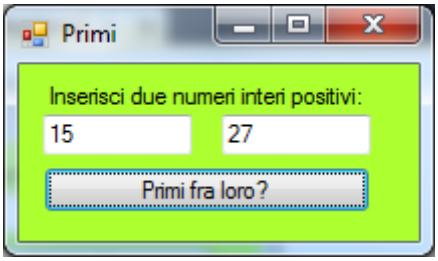
ESERCIZIO 1. NUMERO PRIMO

Definire un metodo che restituisce un valore booleano che indica se un numero passato è primo.

L'utente scrive un numero nella textBox; il pulsante invoca la funzione e poi visualizza un messaggio con l'esito restituito.



ESERCIZIO 2. NUMERI PRIMI FRA LORO



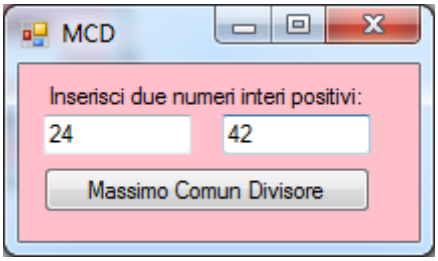
Definire un metodo che restituisce un valore booleano che indica se due numeri sono primi fra loro (hanno solo 1 come divisore comune).

L'utente scrive nelle due textBox dei numeri; il pulsante invoca la funzione e poi visualizza un messaggio con l'esito restituito.

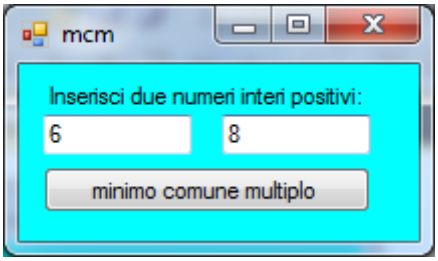
ESERCIZIO 3. MCD

Definire un metodo che calcola il massimo comun divisore tra due numeri interi positivi.

L'utente scrive nelle due textBox dei numeri; il pulsante invoca la funzione e poi visualizza un messaggio con l'esito restituito.



ESERCIZIO 4. MCM



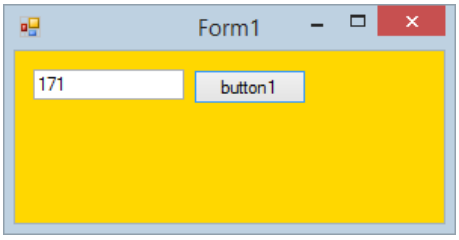
Definire un metodo che calcola il minimo comune multiplo tra due numeri interi positivi.

L'utente scrive nelle due textBox dei numeri; il pulsante invoca la funzione e poi visualizza un messaggio con l'esito restituito.

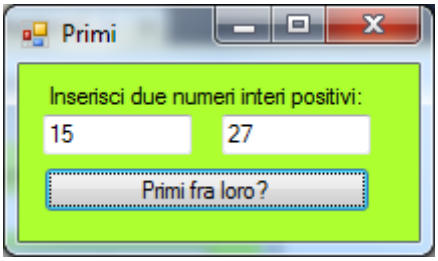
ESERCIZIO 5. PARAMETRI NUMERICI

Definire un metodo che dimezza un valore intero passato per parametro.

L'utente scrive un numero nella textBox1; il pulsante invoca la funzione e poi visualizza nella textBox1 l'esito restituito.



ESERCIZIO 6. PARAMETRI NUMERICI



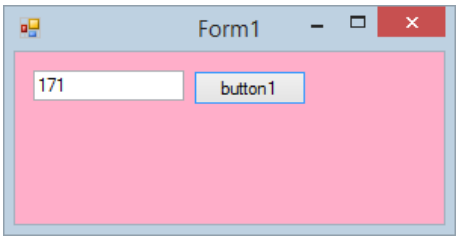
Definire un metodo che incrementa di +1 il minore tra due numeri passati per parametro.

L'utente scrive due numeri nelle textBox; il pulsante invoca la funzione e poi visualizza nelle textBox l'esito del metodo.

ESERCIZIO 7. PARAMETRI NUMERICI

Definire un metodo che se un valore intero passato per parametro è pari lo dimezza, ma se è dispari lo incrementa di +1.

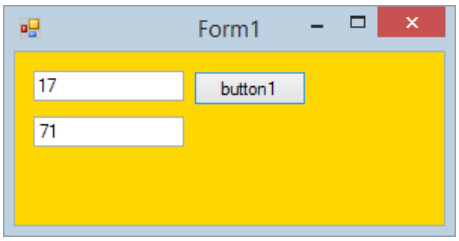
L'utente scrive un numero nella textBox1; il pulsante invoca la funzione e poi visualizza nella textBox1 l'esito restituito.



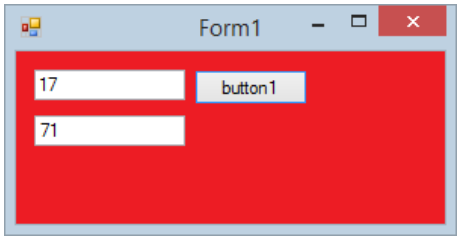
ESERCIZIO 8. PARAMETRI NUMERICI

Definire un metodo che genera due numeri casuali interi compresi tra 10 e 99. I valori devono essere restituiti in parametri passati per risultato.

Il pulsante button1 invoca la funzione e infine mostra i due valori in altrettante textBox.



ESERCIZIO 9. PARAMETRI NUMERICI



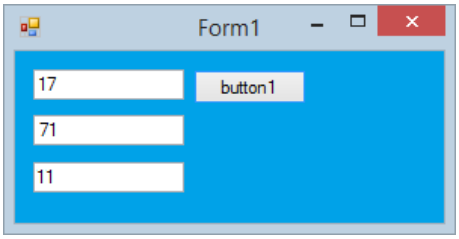
Definire un metodo che genera due numeri casuali interi compresi tra 10 e 99 ma diversi tra loro (evitare numeri identici). I valori devono essere restituiti in parametri passati per risultato.

Il pulsante button1 invoca la funzione e infine mostra i due valori in altrettante textBox.

ESERCIZIO 10. PARAMETRI NUMERICI

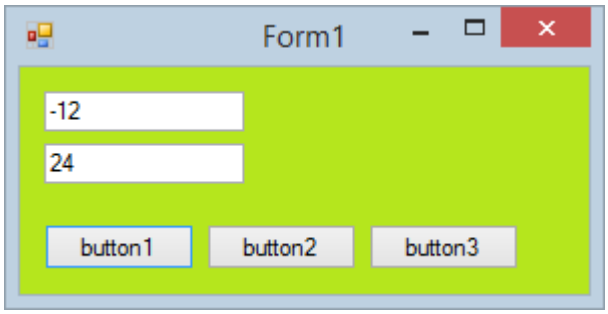
Definire un metodo che genera tre numeri casuali interi compresi tra 10 e 99. I valori devono essere restituiti in parametri passati per risultato.

Il pulsante button1 invoca la funzione e infine mostra i due valori in altrettante textBox.



ESERCIZI SU FUNZIONI CON VETTORI

ESERCIZIO 11. PARAMETRI VETTORI



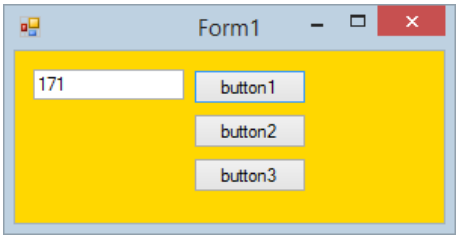
Definire un metodo che riceve come parametro un vettore di interi, e rende (in parametri) i valori rispettivamente minimo e massimo ivi contenuti.

Il pulsante dichiara e prepara un vettore di interi di 100 celle con numeri casuali compresi tra -100 e +100 (inclusi) e poi invoca la funzione e infine visualizza nelle textBox l'esito del metodo

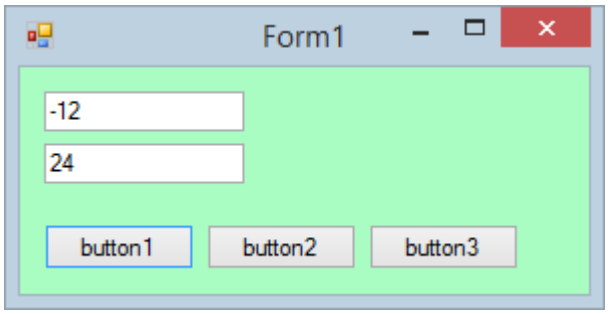
ESERCIZIO 12. PARAMETRI VETTORI

Definire un metodo crea un nuovo vettore di interi istanziato con un numero di celle fissato da un secondo parametro. Il vettore deve essere reso in un parametro.

Il pulsante dichiara un vettore di interi e poi invoca la funzione per costruirlo con 1000 celle e infine visualizza nella textBox1 il contenuto dell'ultima cella.



ESERCIZIO 13. PARAMETRI VETTORI



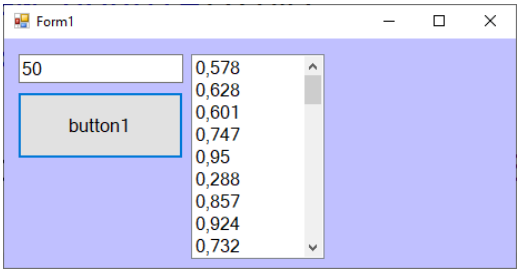
Definire un metodo che riceve come parametro un vettore di interi e due interi che rappresentano il minimo e il massimo dei valori generabili al suo interno; il metodo rende (in un parametro) il vettore con 250 celle già inizializzato.

Il pulsante dichiara un vettore di interi e poi invoca la funzione per costruirlo e infine visualizza nella textBox1 il contenuto dell'ultima cella.

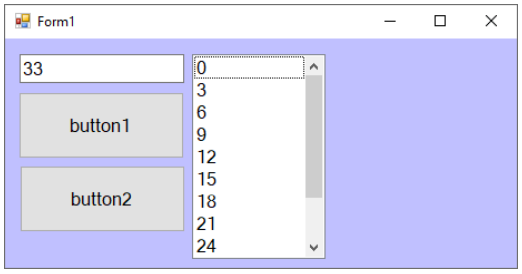
ESERCIZIO 14.

Definire un metodo crea un nuovo vettore di decimali (con la virgola) istanziato con un numero di celle fissato da un parametro. Le celle devono essere inizializzate con valori decimali compresi tra 0 e 1 (estremi a piacere). Il vettore deve essere restituito come tipo reso dal metodo.

Il pulsante dichiara un vettore di interi e poi invoca la funzione per costruirlo con 1000 celle e infine visualizza in una ListBox1 il contenuto delle celle.



ESERCIZIO 15.



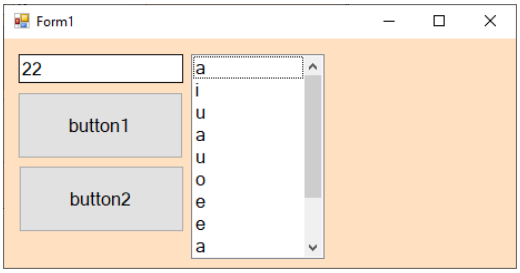
Definire un metodo crea un nuovo vettore di interi istanziato con un numero di celle fissato da un parametro. Le celle devono essere inizializzate con valori identici al triplo del proprio indice. Il vettore deve essere restituito come tipo reso dal metodo.

Il pulsante dichiara un vettore di stringhe e poi invoca la funzione per costruirlo con un numero di celle scelto da una textBox e infine visualizza in una ListBox1 il contenuto delle celle.

ESERCIZIO 16.

Definire un metodo crea un nuovo vettore di stringhe istanziato con un numero di celle fissato da un parametro. Le celle devono essere inizializzate con valori casuali scelti tra le lettere vocali dell'alfabeto. Il vettore deve essere restituito come tipo reso dal metodo.

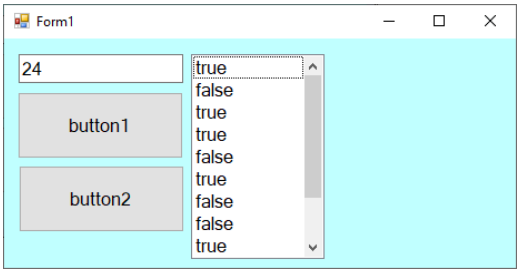
Il pulsante dichiara un vettore e poi invoca la funzione per costruirlo con un numero di celle scelto da una textBox e infine visualizza in una ListBox1 il contenuto delle celle.



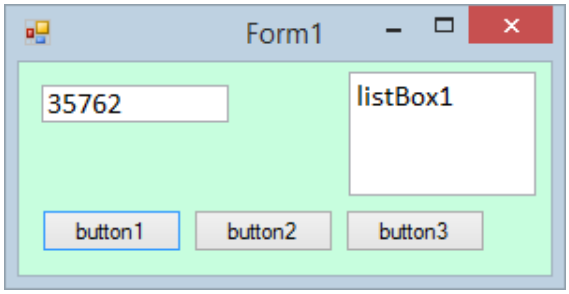
ESERCIZIO 17.

Definire un metodo crea un nuovo vettore di booleani istanziato con un numero di celle fissato da un parametro. Le celle devono essere inizializzate con valori casuali. Il vettore deve essere restituito come tipo reso dal metodo.

Il pulsante dichiara un vettore e poi invoca la funzione per costruirlo con un numero di celle scelto da una textBox e infine visualizza in una ListBox1 il contenuto delle celle.



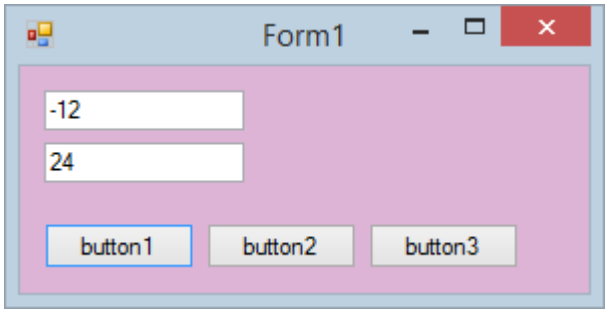
ESERCIZIO 18.



Definire un metodo che riceve come parametro un vettore di interi, e rende il prodotto di tutte le celle.

Il pulsante dichiara e prepara il vettore di 10 celle con valori compresi tra 1 e 9 e poi invoca il metodo e infine mostra in una textBox1 il risultato del metodo.

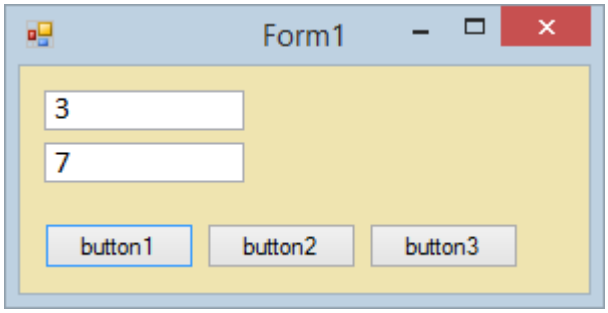
ESERCIZIO 19. FUSIONE DI VETTORI



Definire un metodo **Fusione** che riceve come parametri due vettori di interi e rende un nuovo vettore ottenuto dalla fusione dei primi due.

Il pulsante **button1** invoca la funzione **Fusione** per costruirlo e infine visualizza nella textBox1 il contenuto dell'ultima cella.

ESERCIZIO 20. INDICI DI VETTORI



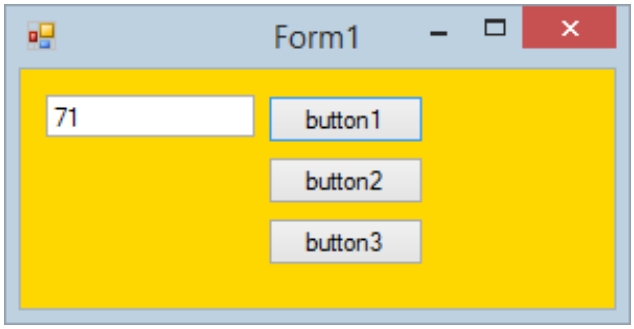
Definire un metodo che riceve come parametro un vettore di interi, e rende (in parametri) i valori rispettivamente degli indici delle celle che contengono il minimo e il massimo contenuti in esso.

Il pulsante dichiara e prepara un vettore di interi di 100 celle con numeri casuali compresi tra -100 e +100 (inclusi) e poi invoca la funzione e infine visualizza nelle textBox l'esito del metodo

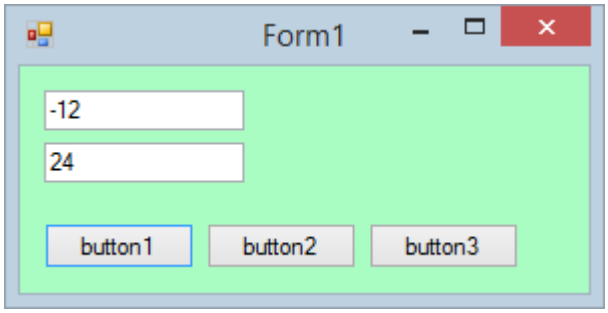
ESERCIZIO 21.

Definire un metodo che crea un nuovo vettore di interi istanziato con un numero di celle fissato da un parametro e che contiene valori casuali compresi tra due estremi scelti con due parametri. Il vettore deve essere reso in un parametro passato per risultato.

Il pulsante dichiara un vettore di interi e poi invoca la funzione per costruirlo con 33 celle e valori compresi tra 10 e 99 e infine visualizza nella textBox1 il contenuto dell'ultima cella.



ESERCIZIO 22.



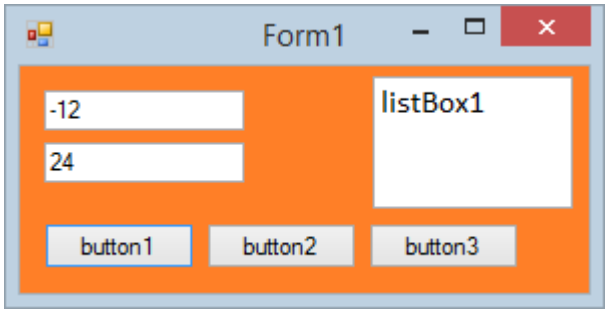
Definire un metodo che riceve come parametri due vettori di interi e cerca i valori minimo e massimo contenuti in essi. Alla fine sono restituiti sempre due valori interi.

Il pulsante dichiara due vettori di interi e li prepara e poi invoca la funzione per effettuare questa ricerca e infine visualizza nelle textBox i risultati trovati.

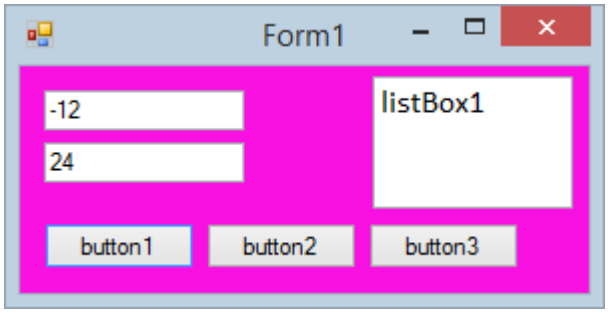
ESERCIZIO 23.

Definire un metodo **Comune** che riceve come parametri due vettori di interi e rende vero (true) se hanno la stessa media, falso (false) altrimenti.

Il pulsante **button1** prepara due vettori e poi invoca la funzione e colora di verde la finestra se il risultato è vero, di rosso altrimenti.



ESERCIZIO 24.



Definire un metodo **Fusione** che riceve come parametri due vettori di interi e rende un nuovo vettore ottenuto dalla fusione dei primi due.

Il pulsante **button1** invoca la funzione **Fusione** per costruirlo e infine visualizza in una listBox il vettore.

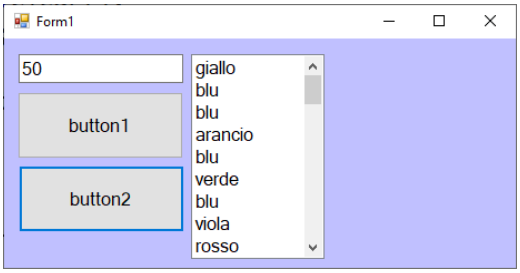
ESERCIZIO 25.

Definire un metodo crea un nuovo vettore di stringhe istanziato con un numero di celle fissato da un parametro. Le celle devono essere inizializzate una delle seguenti stringhe:

arancio, blu, giallo, rosso, verde, viola

Il vettore deve essere restituito come tipo reso dal metodo.

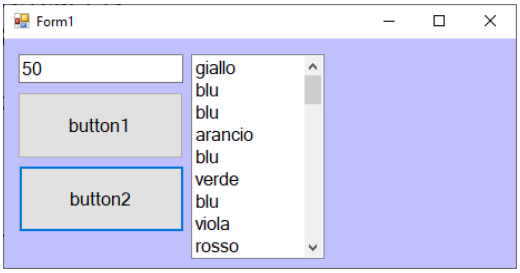
Il pulsante dichiara un vettore e poi invoca la funzione per costruirlo con un numero di celle selezionato da casella di testo e infine visualizza in una ListBox1 il contenuto delle celle.



ESERCIZIO 26.

Dichiarare il seguente metodo:

```
public string[] CreaS(int dim)
{
  string[] tmp = new string[dim];
  string[] colori =
       {"arancio", "blu", "giallo",
        "rosso", "verde", "viola"};
  for (int k = 0; k < dim; k++)
  {
    tmp[k] = colori[random.Next(colori.Length)];
  }
       return tmp;
}
```



e capire cosa fa

SOMMARIO