Le funzioni

parte 2

Passaggi di parametri multipli

La keyword `params` maschera il fatto che la funzione riceva in input un array.
In questo modo il chiamante può scegliere se passare i parametri di input come array o come singoli valori.

```
long SommaNumeri(params int[] numeri)
{
    long somma = 0;
    foreach (int n in numeri)
        somma += n;
    return somma;
}

void Main()
{
    long tot = SommaNumeri(4,3,66,43);
    // ...
}
```

Passaggio per riferimento esplicito per output

La keyword `out` contrassegna un parametro di input come output aggiuntivo della funzione. Questo significa che quel parametro sarà usato dalla funzione come variabile di output per restituire il risultato al chiamante.

```
bool TryParseInt(string s, out int v)
{
    foreach (char c in s)
        if (!char.IsNumber(c))
        {
            v = 0;
            return false;
        }
    v = int.Parse(s);
    return true;
}

void Main()
{
    int n;
    string sn;
    Console.Write("Inserisci un numero intero:");
    sn = Console.ReadLine();
    if(TryParseInt(sn, out n))
        Console.WriteLine("Bravo! Hai inserito il numero intero {0}.", n);
    else
        Console.WriteLine("Attento! Non hai inserito un numero intero.");
}
```

Valorizzare SEMPRE la variabile marcata con **out** prima del return altrimenti il compilatore segnala **errore**!

ref vs out

ref

- la funzione può leggere il valore iniziale della variabile
- la funzione non è obbligata a modificare la variabile

out

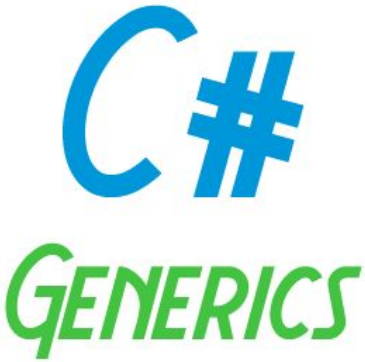
- la funzione non può leggere il valore che ha la variabile
- la funzione è obbligata a dare un valore alla variabile prima del termine

Generics

Tecnica che permette di creare funzioni (e tante altre cose) senza dichiarare il tipo di dato concreto delle variabili su cui la funziona lavora.

Questa tecnica permette di programmare la funzione una sola volta per diversi tipi di dato.

Vediamo un esempio: dobbiamo implementare una funzione che scambia il valore di due variabili.



Generics

Immaginiamo di dover scrivere la funzione che effettua lo scambio di due variabili intere (vi ricordate il BubbleSort???)

Ma se le variabili da scambiare fossero long? e poi double? e poi float? e poi char? e poi string?

Devo riscrivere la stessa funzione per ogni tipologia di tipo di dato?

```
void Scambio(ref int n1, ref int n2)
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}

void Main()
{
    int n1 = 5, n2 = 7;
    Scambio(ref n1, ref n2);
    // ...
}
```

Generics

Sfruttando i generics posso scrivere una funzione che effettua lo scambio di due variabili senza essere a conoscenza del loro tipo di dato.

```
void Scambio<T>(ref T n1, ref T n2)
{
    T temp = n1;
    n1 = n2;
    n2 = temp;
}

void Main()
{
    int n1 = 5, n2 = 7;
    double d1 = 4.56, d2 = 8.99;
    Scambio<int>(ref n1, ref n2);
    Scambio<double>(ref d1, ref d2);
    // ...
}
```

struct e funzioni

Fino ad ora abbiamo visto le **struct** solo come banali **contenitori di dati**.

Una struct è molto di più: può rappresentare un'entità all'interno del software (vedi la struct Libro o la struct Autore)

In questi casi, in quanto **entità**, può essere significativo inserire all'interno di una struct non solo **informazioni** (variabili) ma anche **comportamenti** (funzioni).

struct e funzioni

Ad esempio, possiamo definire una funzione all'interno della struct Libro.

Questa funzione produce un risultato lavorando internamente sui dati del libro.

```
struct Libro
{
    public string Titolo, Autore;
    public int AnnoPubblicazione;
    public uint Pagine;

    public int AnniDiPubblicazione()
    {
        return DateTime.Now.Year - AnnoPubblicazione;
    }
}

void Main()
{
    Libro l = new Libro();
    l.Autore = "Italo Calvino";
    l.Titolo = "Il cavaliere inesistente";
    l.AnnoPubblicazione = 2015;

    int anni = l.AnniDiPubblicazione();

    Console.WriteLine("Il libro è stato pubblicato {0} anni fa.", anni);
}
```

Overloading

Stessa funzione ma diversi parametri di input e di output.

```
int MyMethod(int x)
float MyMethod(float x)
double MyMethod(double x, double y)
```

```
static int PlusMethod(int x, int y)
{
  return x + y;
}

static double PlusMethod(double x, double y)
{
  return x + y;
}

static void Main(string[] args)
{
  int myNum1 = PlusMethod(8, 5);
  double myNum2 = PlusMethod(4.3, 6.26);
  Console.WriteLine("Int: " + myNum1);
  Console.WriteLine("Double: " + myNum2);
}
```