Guida Completa C#

per Principianti

MOD. I – PROGRAMMARE IN C# • 108 ORE

Tipi di Dato • Input/Output • Aritmetica • Operatori • Cicli • Array • List • Metodi • Strutture

1. TIPI DI DATO

>>> Cos'è una Variabile?

Una variabile è un contenitore con un nome che conserva un valore nella memoria del computer. Prima di usarla, devi dichiararla specificando il tipo e il nome.

```
int eta = 25;
// int = tipo | eta = nome | 25 = valore
```

>>> Tipi Numerici Interi (senza decimali)

Tipo	Dimensione	Valore Min	Valore Max	Uso tipico
byte	8 bit	0	255	numeri piccoli positivi
short	16 bit	-32.768	32.767	numeri piccoli
int	32 bit	-2.147.483.648	2.147.483.647	uso generale
long	64 bit	-9.2 quintilioni	9.2 quintilioni	numeri enormi

```
byte piccolo = 200;
short medio = 30000;
int normale = 1000000;
long enorme = 9000000000L;  // la 'L' indica che è un long
```

>>> Tipi Numerici Decimali

Tipo	Precisione	Uso tipico
float	~7 cifre	grafica, fisica
double	~15 cifre	calcoli generali
decimal	~28 cifre	soldi, finanza

```
float temperatura = 36.6f;   // 'f' obbligatorio
double altezza = 1.75;       // default per i decimali
decimal prezzo = 19.99m;     // 'm' obbligatorio, massima precisione
```

⚠NOTE: Per i soldi usa sempre decimal — float e double possono avere errori di arrotondamento!

>>> Altri Tipi Base

```
char lettera = 'A';       // singolo carattere (virgolette singole)
string nome = "Marco";   // testo (virgolette doppie)
bool attivo = true;       // solo true o false
```

>>> Variabili con var (tipo implicito)

C# può dedurre il tipo automaticamente con var:

```
var numero = 42;      // deduce int
var testo = "Ciao";  // deduce string
var prezzo = 9.99;   // deduce double
```

⚠NOTE: Il tipo viene comunque assegnato — non puoi cambiarlo dopo!

>>> Costanti (const)

Una costante ha un valore che non può mai cambiare:

```
const double PI = 3.14159;
const int MAX_STUDENTI = 30;
// PI = 3.0; // ERRORE! Non si può modificare
```

>>> Conversione di Tipo (Type Casting)

```
// Implicita (automatica, nessun rischio di perdita dati):
int numero = 10;
double risultato = numero;   // int → double, automatico

// Esplicita (manuale, possibile perdita di dati):
double d = 9.99;
int i = (int)d;   // i = 9, il decimale viene perso!

// Con metodi di conversione:
string testo = "42";
int n = int.Parse(testo);          // stringa → int
double dd = double.Parse("3.14"); // stringa → double
int nn = Convert.ToInt32("100");  // alternativa con Convert
string s = Convert.ToString(99);  // numero → stringa
```

>>> Overflow

Se superi il valore massimo di un tipo, si verifica un overflow:

```
byte b = 255;
b++;  // b diventa 0! (torna all'inizio)
```

2. VISUALIZZAZIONE E ACQUISIZIONE DATI

>>> Output — Mostrare Dati a Schermo

```
Console.WriteLine("Ciao!");  // stampa e va a capo
Console.Write("Ciao ");      // stampa SENZA andare a capo
Console.WriteLine();          // riga vuota

// Concatenazione con +:
string nome = "Marco"; int eta = 25;
Console.WriteLine("Nome: " + nome + ", Età: " + eta);

// String Interpolation (modo moderno e preferito):
Console.WriteLine($"Nome: {nome}, Età: {eta}");

// Formattazione numeri:
double prezzo = 1234.5678;
Console.WriteLine(prezzo.ToString("F2"));  // 1234,57 (2 decimali)
Console.WriteLine(prezzo.ToString("C"));   // €1.234,57 (valuta)
Console.WriteLine(prezzo.ToString("N0"));  // 1.235 (nessun decimale)

// Caratteri speciali:
Console.WriteLine("Riga 1\nRiga 2");       // \n = nuova riga
Console.WriteLine("Col1\tCol2");           // \t = tab
```

>>> Input — Leggere Dati dall'Utente

```
Console.Write("Inserisci il tuo nome: ");
string nome = Console.ReadLine();   // legge una riga di testo

Console.Write("Inserisci la tua età: ");
int eta = int.Parse(Console.ReadLine());   // converte subito in int

// Metodo sicuro con TryParse:
string input = Console.ReadLine();
int numero;
if (int.TryParse(input, out numero))
    Console.WriteLine($"Hai inserito: {numero}");
else
    Console.WriteLine("Errore: non è un numero valido!");
```

>>> Metodi Utili per le Stringhe

```
string testo = " Ciao Mondo ";

testo.Length                    // lunghezza: 12
testo.ToUpper()                 // " CIAO MONDO "
testo.ToLower()                 // " ciao mondo "
testo.Trim()                    // "Ciao Mondo" (rimuove spazi)
testo.Replace("Mondo","Italia") // " Ciao Italia "
testo.Contains("Ciao")          // true
testo.Substring(1, 4)           // "Ciao"
testo.Split(' ')                // divide in array per spazio
testo.IndexOf("Mondo")          // posizione della parola
```

3. CALCOLI ARITMETICI

>>> Operatori Base

```
int a = 10, b = 3;

int somma      = a + b;   // 13
int differenza = a - b;   // 7
int prodotto   = a * b;   // 30
int quoziente  = a / b;   // 3  (attenzione! int/int = int)
int resto      = a % b;   // 1  (modulo = resto della divisione)
```

⚠NOTE: 10 / 3 = 3 e non 3.33 perché sono entrambi int! Per avere i decimali: 10.0 / 3 oppure (double)10 / 3

>>> Operatori di Assegnazione Composta

```
int n = 10;
n += 5;   // n = n + 5  → 15
n -= 3;   // n = n - 3  → 12
n *= 2;   // n = n * 2  → 24
n /= 4;   // n = n / 4  → 6
n %= 4;   // n = n % 4  → 2
```

>>> Incremento e Decremento

```
int n = 5;
n++;   // post-incremento: usa n poi aggiunge 1
++n;   // pre-incremento: aggiunge 1 poi usa n
n--;   // post-decremento
--n;   // pre-decremento

// Differenza tra pre e post:
int a = 5;
int b = a++;   // b = 5, poi a diventa 6
int c = ++a;   // a diventa 7, poi c = 7
```

>>> Libreria Math

```
Math.Abs(-5)          // 5   (valore assoluto)
Math.Sqrt(16)         // 4.0 (radice quadrata)
Math.Pow(2, 10)       // 1024.0 (potenza: 2^10)
Math.Round(3.567, 2)  // 3.57 (arrotonda a 2 decimali)
Math.Floor(3.9)       // 3.0  (arrotonda per difetto)
Math.Ceiling(3.1)     // 4.0  (arrotonda per eccesso)
Math.Max(10, 20)      // 20   (il maggiore dei due)
Math.Min(10, 20)      // 10   (il minore dei due)
Math.PI               // 3.14159...
```

4. OPERATORI DI CONFRONTO

Confrontano due valori e restituiscono sempre true o false.

```
int a = 10, b = 5;

a == b   // false — uguale a
a != b   // true  — diverso da
a >  b   // true  — maggiore di
a <  b   // false — minore di
a >= b   // true  — maggiore o uguale a
a <= b   // false — minore o uguale a
```

⚠ NOTE: Non confondere = (assegnazione) con == (confronto)! → a = 5 assegna, a == 5 controlla.

5. OPERATORI BOOLEANI E LOGICI

>>> Operatore AND (&&)

Il risultato è true solo se entrambe le condizioni sono vere.

```
bool maggiorenne = eta >= 18;
bool haDocumento = true;

if (maggiorenne && haDocumento)
    Console.WriteLine("Accesso consentito");
```

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

>>> Operatore OR (||)

Il risultato è true se almeno una condizione è vera.

```
bool haCarta = false;
bool haContanti = true;

if (haCarta || haContanti)
    Console.WriteLine("Può pagare");
```

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

>>> Operatore NOT (!)

Inverte il valore booleano.

```
bool connesso = false;

if (!connesso)
    Console.WriteLine("Non sei connesso!");

// !true = false
// !false = true
```

>>> Operatore Ternario

Un if/else compatto su una riga:

```
// condizione ? valoreSeTrue : valoreSefalse
string risultato = eta >= 18 ? "Adulto" : "Minore";
Console.WriteLine(risultato);
```

6. STRUTTURE DI CONTROLLO

>>> if / else if / else

```
int voto = 75;

if (voto >= 90)
{
    Console.WriteLine("Ottimo!");
}
else if (voto >= 75)
{
    Console.WriteLine("Buono");
}
else if (voto >= 60)
{
    Console.WriteLine("Sufficiente");
}
else
{
    Console.WriteLine("Insufficiente");
}
```

>>> switch

```
int giorno = 2;

switch (giorno)
{
    case 1:
        Console.WriteLine("Lunedì");
        break;
    case 2:
        Console.WriteLine("Martedì");
        break;
    case 6:
    case 7:
        Console.WriteLine("Weekend!");  // più case per lo stesso blocco
        break;
    default:
        Console.WriteLine("Altro giorno");
        break;
}
```

⚠NOTE: Il break è obbligatorio alla fine di ogni case!

>>> switch expression (moderno, C# 8+)

```
string nomeGiorno = giorno switch
{
    1 => "Lunedì",
    2 => "Martedì",
    3 => "Mercoledì",
    _ => "Altro"   // _ è il default
};
```

7. CICLO FOR

Usato quando sai esattamente quante volte ripetere.

```
// Struttura:
for (inizializzazione; condizione; aggiornamento)
{
    // codice da ripetere
}

// Esempio base:
for (int i = 0; i < 5; i++)
{
    Console.WriteLine($"Iterazione: {i}");
}
// stampa 0, 1, 2, 3, 4

// Contare al contrario:
for (int i = 10; i >= 0; i--)
    Console.Write(i + " ");
// 10 9 8 7 6 5 4 3 2 1 0

// Saltare elementi (step diverso da 1):
for (int i = 0; i <= 20; i += 2)
    Console.Write(i + " ");
// 0 2 4 6 8 10 12 14 16 18 20

// For annidati:
for (int riga = 1; riga <= 3; riga++)
{
    for (int col = 1; col <= 3; col++)
        Console.Write($"[{riga},{col}] ");
    Console.WriteLine();
}
// [1,1] [1,2] [1,3]
// [2,1] [2,2] [2,3]
// [3,1] [3,2] [3,3]

// break e continue:
for (int i = 0; i < 10; i++)
{
    if (i == 3) continue;   // salta il 3
    if (i == 7) break;      // esce dal ciclo
    Console.Write(i + " ");
}
// 0 1 2 4 5 6
```

8. CICLO WHILE

Ripete finché la condizione è vera. Utile quando non sai in anticipo quante iterazioni fare. La condizione viene verificata PRIMA di ogni iterazione.

```
// Struttura:
while (condizione)
{
    // codice da ripetere
}

// Esempio base:
int i = 0;
while (i < 5)
{
    Console.WriteLine($"i = {i}");
    i++;
}

// Esempio pratico — indovina il numero:
int numeroDaIndovinare = 7;
int tentativo = 0;

while (tentativo != numeroDaIndovinare)
{
    Console.Write("Indovina il numero: ");
    tentativo = int.Parse(Console.ReadLine());

    if (tentativo < numeroDaIndovinare)
        Console.WriteLine("Troppo basso!");
    else if (tentativo > numeroDaIndovinare)
        Console.WriteLine("Troppo alto!");
}
Console.WriteLine("Hai indovinato!");
```

⚠ NOTE: Se la condizione non diventa mai falsa → loop infinito! Assicurati sempre che qualcosa cambi ad ogni iterazione.

9. CICLO DO WHILE

Come il while, ma la condizione viene verificata DOPO l'esecuzione del blocco. Il codice viene eseguito almeno una volta sempre.

```
// Struttura:
do
{
    // codice da ripetere
} while (condizione);

// Esempio — menu ripetuto:
string scelta;

do
{
    Console.WriteLine("=== MENU ===");
    Console.WriteLine("1. Gioca");
    Console.WriteLine("2. Impostazioni");
    Console.WriteLine("0. Esci");
    Console.Write("Scelta: ");
    scelta = Console.ReadLine();

    if (scelta == "1") Console.WriteLine("Stai giocando...");
    if (scelta == "2") Console.WriteLine("Impostazioni...");

} while (scelta != "0");

Console.WriteLine("Arrivederci!");
```

>>> Confronto tra i tre cicli

	for	while	do while
Quando usarlo	Iterazioni note	Iterazioni non note	Almeno 1 esecuzione garantita
Condizione verificata	Prima	Prima	Dopo
Esecuzioni minime	0	0	1
Esempio tipico	Scorrere array	Input utente valido	Menu

10. ARRAY E CICLO FOREACH

>>> Cos'è un Array?

Un array è una collezione ordinata di elementi dello stesso tipo, con dimensione fissa.

```
// Indice: [0]  [1]  [2]  [3]  [4]
// Valori:  10   20   30   40   50
```

⚠NOTE: L'indice parte sempre da 0, non da 1!

>>> Dichiarazione e Inizializzazione

```
// Metodo 1: dichiarazione con dimensione
int[] numeri = new int[5];   // 5 elementi, tutti 0
numeri[0] = 10;
numeri[1] = 20;

// Metodo 2: dichiarazione con valori
int[] numeri2 = { 10, 20, 30, 40, 50 };

// Metodo 3: new con valori
string[] frutti = new string[] { "Mela", "Banana", "Ciliegia" };

// Accesso:
Console.WriteLine(numeri2[0]);     // 10
Console.WriteLine(numeri2[4]);     // 50
Console.WriteLine(numeri2.Length); // 5
```

>>> Ciclo foreach

Più semplice e leggibile, ideale per leggere tutti gli elementi:

```
string[] frutti = { "Mela", "Banana", "Ciliegia", "Uva" };

foreach (string frutto in frutti)
{
    Console.WriteLine(frutto);
}
```

⚠NOTE: Con foreach non puoi modificare gli elementi dell'array né accedere all'indice direttamente.

>>> Array Multidimensionale (2D)

Come una tabella con righe e colonne:

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

// Scorrere array 2D:
for (int r = 0; r < griglia.GetLength(0); r++)
{
    for (int c = 0; c < griglia.GetLength(1); c++)
        Console.Write(griglia[r, c] + " ");
    Console.WriteLine();
}
// GetLength(0) = numero di righe
// GetLength(1) = numero di colonne
```

>>> Metodi Utili per gli Array

```
int[] numeri = { 5, 2, 8, 1, 9, 3 };

Array.Sort(numeri);                          // ordina: { 1, 2, 3, 5, 8, 9 }
Array.Reverse(numeri);                       // inverte: { 9, 8, 5, 3, 2, 1 }
int pos = Array.IndexOf(numeri, 5);          // trova l'indice del valore 5
Array.Clear(numeri, 0, numeri.Length);       // azzera tutti gli elementi
int[] copia = new int[numeri.Length];
Array.Copy(numeri, copia, numeri.Length);  // copia array
```

11. ARRAY DINAMICI — List<T>

Gli array normali hanno dimensione fissa. List<T> cresce e diminuisce dinamicamente.

```
using System.Collections.Generic;   // da aggiungere in cima al file

List<int> numeri    = new List<int>();
List<string> nomi   = new List<string>();
List<double> prezzi = new List<double> { 9.99, 4.50, 12.00 };
```

>>> Operazioni Principali

```
List<string> nomi = new List<string>();

// Aggiungere:
nomi.Add("Marco");
nomi.Add("Giulia");
nomi.Insert(1, "Anna");   // inserisce all'indice 1

// Accesso:
Console.WriteLine(nomi[0]);     // Marco
Console.WriteLine(nomi.Count); // 3  (usa Count, non Length!)

// Verificare / cercare:
bool c   = nomi.Contains("Giulia");  // true
int idx  = nomi.IndexOf("Giulia");   // 2

// Rimuovere:
nomi.Remove("Luca");    // rimuove per valore
nomi.RemoveAt(0);        // rimuove per indice

// Ordinare / svuotare:
nomi.Sort();
nomi.Clear();

// Controllare se vuota:
bool vuota = nomi.Count == 0;   // true
```

>>> Scorrere una List

```
List<string> frutti = new List<string> { "Mela", "Banana", "Uva" };

// Con foreach (preferito):
foreach (string frutto in frutti)
    Console.WriteLine(frutto);

// Con for:
for (int i = 0; i < frutti.Count; i++)
    Console.WriteLine($"{i}: {frutti[i]}");
```

>>> Confronto Array vs List<T>

	Array	List<T>
Dimensione	Fissa	Dinamica
Dichiarazione	int[] arr = new int[5]	List<int> lst = new List<int>()
Lunghezza	.Length	.Count

Aggiungere	impossibile	.Add()
Rimuovere	impossibile	.Remove()
Quando usarlo	Dimensione nota e fissa	Dimensione variabile

12. I METODI

>>> Cos'è un Metodo?

Un metodo è un blocco di codice con un nome, che esegue un compito specifico. Invece di scrivere lo stesso codice più volte, lo metti in un metodo e lo chiami quando serve.

>>> Struttura di un Metodo

```
// modificatore  tipoRitorno  nome  (parametri)
static void StampaSaluto()
{
    Console.WriteLine("Ciao!");
}

// Chiamata nel Main:
StampaSaluto();
```

Parte	Significato
static	Appartiene alla classe, non a un oggetto
void	Non restituisce nessun valore
StampaSaluto	Nome del metodo (scelto da te)
()	Parentesi per i parametri (vuote = nessun parametro)
{ }	Corpo del metodo — il codice che esegue

>>> Metodo con Parametri

I parametri sono i valori che passi al metodo quando lo chiami.

```
static void Saluta(string nome)
{
    Console.WriteLine($"Ciao, {nome}!");
}

Saluta("Marco");    // Ciao, Marco!
Saluta("Giulia");   // Ciao, Giulia!

// Più parametri:
static void StampaSomma(int a, int b)
{
    Console.WriteLine($"{a} + {b} = {a + b}");
}
StampaSomma(10, 5);   // 10 + 5 = 15
```

>>> Metodo con Valore di Ritorno (return)

Se il metodo deve restituire un risultato, usa un tipo al posto di void e aggiungi return.

```
static int Somma(int a, int b)
{
    return a + b;
}

int risultato = Somma(10, 5);
Console.WriteLine(risultato);       // 15
```

```
Console.WriteLine(Somma(3, 7));     // 10

// Esempi con tipi diversi:
static double CalcolaMedia(int[] valori)
{
    int somma = 0;
    foreach (int v in valori) somma += v;
    return (double)somma / valori.Length;
}

static bool IsMaggiore(int n)  { return n >= 18; }

static string Giudizio(int voto)
{
    if (voto >= 90) return "Ottimo";
    else if (voto >= 75) return "Buono";
    else if (voto >= 60) return "Sufficiente";
    else return "Insufficiente";
}
```

>>> Parametri Opzionali (Valori Default)

```
static void Saluta(string nome, string saluto = "Ciao")
{
    Console.WriteLine($"{saluto}, {nome}!");
}

Saluta("Marco");                // Ciao, Marco!
Saluta("Giulia", "Buongiorno"); // Buongiorno, Giulia!
```

⚠NOTE: I parametri opzionali devono sempre essere gli ultimi nella lista!

>>> Overloading — Stesso Nome, Parametri Diversi

Puoi creare più metodi con lo stesso nome ma parametri diversi — C# capisce quale usare in base agli argomenti.

```
static void Stampa(int numero)    { Console.WriteLine($"Intero: {numero}");
}
static void Stampa(double numero) { Console.WriteLine($"Decimale:
{numero}"); }
static void Stampa(string testo)  { Console.WriteLine($"Testo: {testo}"); }

Stampa(42);       // Intero: 42
Stampa(3.14);     // Decimale: 3.14
Stampa("Ciao");   // Testo: Ciao
```

>>> Passaggio per Valore vs per Riferimento

```
// Per VALORE (default) — il metodo riceve una copia:
static void Raddoppia(int n) { n = n * 2; }

int x = 5;
Raddoppia(x);
Console.WriteLine(x);  // ancora 5! (la copia non influisce su x)

// Per RIFERIMENTO con ref — modifica la variabile originale:
static void RaddoppiaRef(ref int n) { n = n * 2; }

int y = 5;
RaddoppiaRef(ref y);
```

```
Console.WriteLine(y);  // 10!
```

>>> Parametro out

out è simile a ref, ma il metodo DEVE assegnare un valore. Usato per restituire più valori.

```
static void CalcolaMinMax(int[] arr, out int min, out int max)
{
    min = arr[0]; max = arr[0];
    foreach (int n in arr)
    {
        if (n < min) min = n;
        if (n > max) max = n;
    }
}

int[] numeri = { 5, 2, 9, 1, 7 };
CalcolaMinMax(numeri, out int minimo, out int massimo);
Console.WriteLine($"Min: {minimo}, Max: {massimo}");
// Min: 1, Max: 9
```

>>> Metodi Ricorsivi

Un metodo ricorsivo chiama se stesso. Deve avere sempre una condizione di uscita (caso base).

```
static int Fattoriale(int n)
{
    if (n <= 1) return 1;          // caso base
    return n * Fattoriale(n - 1); // chiamata ricorsiva
}

Console.WriteLine(Fattoriale(5));   // 5*4*3*2*1 = 120
```

⚠NOTE: Senza il caso base, la ricorsione è infinita e causa un StackOverflowException!

13. LE STRUTTURE (struct)

>>> Cos'è una Struttura?

Una struttura (struct) è un tipo di dato personalizzato che raggruppa più variabili correlate sotto un unico nome. È simile a una classe ma più leggera — ideale per rappresentare dati semplici come un Punto (x,y), una Data, un Prodotto.

>>> Dichiarazione di una Struct

```
struct Punto
{
    public int X;
    public int Y;
}

// Utilizzo:
Punto p;
p.X = 10;
p.Y = 20;
Console.WriteLine($"Punto: ({p.X}, {p.Y})");
// Punto: (10, 20)
```

>>> Costruttore nella Struct

Puoi aggiungere un costruttore per inizializzare tutti i campi in una sola riga.

```
struct Prodotto
{
    public string Nome;
    public double Prezzo;
    public int Quantita;

    public Prodotto(string nome, double prezzo, int quantita)
    {
        Nome     = nome;
        Prezzo   = prezzo;
        Quantita = quantita;
    }
}

// Utilizzo con costruttore:
Prodotto p = new Prodotto("Pizza", 8.50, 10);
Console.WriteLine($"{p.Nome}: {p.Prezzo}€ x{p.Quantita}");
// Pizza: 8.5€ x10
```

>>> Metodi nella Struct

```
struct Rettangolo
{
    public double Larghezza;
    public double Altezza;

    public Rettangolo(double l, double a) { Larghezza = l; Altezza = a; }

    public double Area()       { return Larghezza * Altezza; }
    public double Perimetro()  { return 2 * (Larghezza + Altezza); }

    public void StampaInfo()
    {
```

```
        Console.WriteLine($"Rettangolo {Larghezza}x{Altezza}");
        Console.WriteLine($"Area: {Area()}  Perimetro: {Perimetro()}");
    }
}

Rettangolo r = new Rettangolo(5, 3);
r.StampaInfo();
// Rettangolo 5x3
// Area: 15  Perimetro: 16
```

>>> Array di Strutture

```
struct Studente
{
    public string Nome;
    public int[] Voti;

    public Studente(string nome, int[] voti) { Nome = nome; Voti = voti; }

    public double CalcolaMedia()
    {
        int somma = 0;
        foreach (int v in Voti) somma += v;
        return (double)somma / Voti.Length;
    }
}

Studente[] classe = {
    new Studente("Ilaria",  new int[] { 85, 90, 78 }),
    new Studente("Giorgio", new int[] { 70, 65, 80 }),
    new Studente("Norma",   new int[] { 92, 88, 95 })
};

foreach (Studente s in classe)
    Console.WriteLine($"{s.Nome}: media {s.CalcolaMedia():F1}");
// Ilaria: media 84.3
// Giorgio: media 71.7
// Norma: media 91.7
```

>>> struct vs class — Differenze

Caratteristica	struct	class
Tipo	Valore (Value Type)	Riferimento (Reference Type)
Memoria	Stack	Heap
Ereditarietà	Non supportata	Supportata
Quando usarla	Dati semplici, piccoli	Oggetti complessi
Copia	Copia indipendente	Riferimento condiviso

⚠NOTE: Quando assegni una struct a un'altra variabile, viene copiato il contenuto, non il riferimento!

RIEPILOGO FINALE — MOD. I COMPLETO

#	Argomento	Concetto Chiave
1	Tipi di dato	Definiscono cosa può contenere una variabile
2	Input / Output	Console.ReadLine(), Console.WriteLine()
3	Aritmetica	Operatori + - * / % e compound +=, -=
4	Confronto	== != > < >= <= restituiscono true/false
5	Logici (&&, ||, !)	Combinare condizioni booleane
6	Strutture if / switch	Prendere decisioni nel codice
7	Ciclo for	Ripetere un numero noto di volte
8	Ciclo while	Ripetere finché una condizione è vera
9	Ciclo do while	Come while, ma esegue almeno una volta
10	Array + foreach	Collezione fissa, indice da 0
11	List<T>	Array dinamico che cresce e diminuisce
12	I Metodi	Blocchi di codice riutilizzabili con nome
13	Le Strutture	Tipi personalizzati che raggruppano dati correlati

Fine Modulo I — Programmare in C# (108 ore)