CHAR

La parola chiave **char** è utilizzata per dichiarare un carattere Unicode. I caratteri Unicode sono caratteri a 16 bit utilizzati per rappresentare gran parte delle lingue scritte conosciute. Le costanti del tipo **char** possono essere scritte come caratteri, sequenze esadecimali o rappresentazioni Unicode

char char1 = 'Z'; // Carattere

char char2 = '\x0058'; // Codice in esadecimale da ricercare nella ascii

char char3 = (char)88; // Codice già convertito da cercare nella ascii

char char4 = '\u0058'; // Codice ricercato nella tab unicode

Un oggetto String è un insieme sequenziale di strutture **Char** che rappresenta una stringa di testo. La maggior parte dei caratteri Unicode può essere rappresentata da un singolo oggetto **Char**.

#### Funzionalità

#### La struttura Char fornisce metodi per confrontare gli oggetti Char, convertire il valore dell'oggetto Char corrente su un oggetto di un altro tipo e determinare la categoria Unicode di un oggetto Char

Utilizzare i metodi CompareTo e Equals per confrontare gli oggetti **Char**.

Utilizzare i metodi IsControl, IsDigit, IsLetter, IsLetterOrDigit, IsLower, IsNumber, IsPunctuation, IsSeparator, IsSymbol, IsUpper e IsWhiteSpace per determinare se un carattere appartiene a una categoria Unicode specifica, come ad esempio cifra, lettera, punteggiatura, carattere di controllo e così via.

Utilizzare i metodi GetNumericValue per convertire un oggetto Char che rappresenta un numero in un tipo di valore numerico. Utilizzare i metodi Parse e TryParse per convertire un carattere appartenente a una stringa in un oggetto Char. Utilizzare il metodo ToString per convertire un oggetto Char in un oggetto String.

Utilizzare i metodi ToLower e ToUpper per trasformare da maiuscoli a minuscoli o viceversa i caratteri di un oggetto Char.

Metodo Char.IsControl (Char)

Indica se il carattere Unicode specificato è stato categorizzato come carattere di controllo.

Metodo Char.IsSymbol (Char)

Indica se il carattere Unicode specificato è stato categorizzato come carattere simbolo.

**FOR**

Il ciclo **for** esegue ripetutamente un'istruzione o un blocco di istruzioni finché un'espressione specificata non restituisce **false**. Il ciclo **for** è utile per scorrere le matrici e per l'elaborazione sequenziale. Nell'esempio riportato di seguito il valore di int i viene scritto nella console e i viene incrementato ogni volta di 1 attraverso il ciclo.

|  |
| --- |
| * Innanzitutto, viene valutato il valore iniziale della variabile i. * Quindi, se il valore di i è inferiore a 5, la condizione restituisce **true**, viene eseguita l'istruzione Console.WriteLine e viene rivalutata la variabile i. * Quando i è maggiore di 5, la condizione diventa **false** e il controllo viene trasferito fuori dal ciclo. |

**for** (int i = 1; i <= 5; i++)

{

Console.WriteLine(i);

}

BREAK L'istruzione **break** termina il ciclo di inclusione più vicino o l'istruzione [switch](ms-help://MS.VSCC.v80/MS.MSDN.v80/MS.VisualStudio.v80.it/dv_csref/html/44bae8b8-8841-4d85-826b-8a94277daecb.htm) in cui è contenuta. Il controllo viene passato all'istruzione che segue l'istruzione terminata, se presente.

CONTINUE

|  |
| --- |
| L'istruzione **continue** passa il controllo all'iterazione successiva dell'istruzione di inclusione in cui è presente.  Nell'esempio riportato di seguito viene inizializzato un contatore per il conteggio da 1 a 10. Utilizzando l'istruzione **continue** con l'espressione (i < 9), le istruzioni comprese tra **continue** e la fine del corpo di for verranno ignorate. |

for (int i = 1; i <= 10; i++)

{

if (i < 9)

{

**continue**;

}

Console.WriteLine(i);

}

Output 9-10

**WHILE**

|  |
| --- |
| Output Current value of n is 1  Current value of n is 2  Current value of n is 3  Current value of n is 4  Current value of n is 5 |

L'istruzione **while** esegue un'istruzione o un blocco di istruzioni finché un'espressione specificata non restituisce **false**.

int n = 1;

**while (n < 6)**

{

Console.WriteLine("Current value of n is {0}", n);

n++;

}

DO

L'istruzione **do** esegue ripetutamente un'istruzione o un blocco di istruzioni racchiuse tra **{}** finché un'espressione specificata non restituisce **false**. Nell'esempio riportato di seguito le istruzioni del ciclo **do** vengono eseguite purché la variabile y sia inferiore a 5.

|  |
| --- |
| Output 0  1  2  3  4 |

|  |
| --- |
| A differenza dell'istruzione while, il ciclo del corpo dell'istruzione **do** viene eseguito almeno una volta, indipendentemente dal valore dell'espressione. |

int x = 0;

**do**

{

Console.WriteLine(x);

|  |
| --- |
| Il ciclo while controlla anche il valore iniziale dell espressione mentre il do lo esegue anche almeno 1 volta |

x++;

}

while (x < 5);

----------------------------------------------------------------------------------------------------

CICLO DO-WHILE

Posso usa anche insieme il ciclo do-while

do {

<istruzione>;

} while (<condizione>);

in modo che arrivati ad una certa condizione l utente esce dal ciclo

|  |
| --- |
| **Metodo Pubblico: Math.Floor**  Sottoposto a overload. Restituisce il valore integer massimo minore del o uguale al numero in virgola mobile e precisione doppia specificato.  double a;  a = 10.5  Console.WriteLine(Math.Floor(a));//10 |

CLASSE MATH

|  |
| --- |
| **Campo Math.PI**  Rappresenta il rapporto tra la circonferenza del cerchio e il suo diametro, specificato dalla costante, *π*.  Console.WriteLine("Il Pi greco vale: {0}",Math.PI); //3,14159265358979323846 |

|  |
| --- |
| **Metodo Pubblico: Math.Abs**  Sottoposto a overload. Restituisce il valore assoluto del numero specificato.  double a;  a = -5.16;  Console.WriteLine("Il valore assoluto di {0} è: {1}",a,Math.Abs(a));//5,16 |

|  |
| --- |
| **Metodo Pubblico: Math.Ceiling**  Sottoposto a overload. Restituisce il valore integer minimo maggiore del o uguale al numero a virgola mobile e precisione doppia specificato.  double a;  a = 10.5;  Console.WriteLine(Math.Ceiling(a));//11 |

|  |
| --- |
| **Metodo Pubblico: Math.Max**  Sottoposto a overload. Restituisce il più elevato tra due numeri specificati.  int a, b;  a = 2;  b=10;  Console.WriteLine(Math.Max(a,b));//10  **Metodo Pubblico: Math.Min**  Sottoposto a overload. Restituisce il meno elevato tra due numeri.  int a, b,resto;  a = 2;  b=10;  Console.WriteLine(Math.Min(a,b));//2 |

|  |
| --- |
| **Metodo Math.DivRem**  Calcola il quoziente di due numeri e restituisce il resto come parametro di output.  int Resto;  int Risultato;  Risultato = Math.DivRem(5, 2, **out** Resto);  Console.WriteLine("{0}/{1} = {2}, con il resto di {3}.", 5, 2, Risultato, Resto);  output  This example produces the following results:  5/2 = 2, con il resto di 1. |

|  |
| --- |
| **Metodo Math.Pow** Parametri x Numero in virgola mobile e precisione doppia da elevare a potenza.  Y Numero in virgola mobile e precisione doppia che specifica una potenza. Valore restituito Numero x elevato alla potenza y.  int Base = 5, Potenza = 3;  Console.WriteLine("{0} elevato a {1}: {2}", Base, Potenza, Math.Pow(Base, Potenza));  output  This example produces the following results:  5 elevato a 3: 125 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Math.Sign**  Sottoposto a overload. Restituisce un valore che indica il segno di un numero.   |  |  | | --- | --- | | **Numero** | **Descrizione** | | -1 | value è minore di zero. | | 0 | value è uguale a zero. | | 1 | value è maggiore di zero. | |

|  |
| --- |
| **Metodo Math.Round**  Arrotonda un valore in virgola mobile e precisione doppia alla precisione specificata.  Math.Round(4.4); //Returns 4.0.  Math.Round(4.5); //Returns 4.0.  Math.Round(4.6); //Returns 5.0.  Math.Round(3.44, 1); //Returns 3.4.  Math.Round(3.45, 1); //Returns 3.4.  Math.Round(3.46, 1); //Returns 3.5. |

|  |
| --- |
| **Metodo Pubblico: Math.Sqrt**  Restituisce la radice quadrata del numero specificato  int a = 81;  Console.WriteLine(Math.Sqrt(a));//9 |

|  |
| --- |
| **Math.Truncate**  Sottoposto a overload. Calcola la parte integrale di un numero.  double a, b, c, d;  a = -1.9595;  b = 0.89;  c = 5.36;  d = -98.7;  Console.WriteLine(Math.Truncate(a));//-1  Console.WriteLine(Math.Truncate(b));//0  Console.WriteLine(Math.Truncate(c));//5  Console.WriteLine(Math.Truncate(d));//-98 |

TIPI DI DOUBLE

**Double.Epsilon**

Rappresenta il valore Double positivo più piccolo maggiore di zero. Il valore di questa costante è 4,94065645841247e-324.

**Double.MaxValue**

Rappresenta il valore massimo possibile di Double. Il valore di questa costante è 1,79769313486232e308 positivo.

**Double.MinValue**

Rappresenta il valore minimo possibile di Double. Il valore di questa costante è 1,79769313486232e308 negativo.

**Double.NaN**

Rappresenta un valore che non è un numero (**NaN**). Il valore di questa costante è il risultato della divisione di zero per zero.

Questa costante viene restituita quando il risultato di un'operazione non è definito.

Utilizzare [IsNaN](ms-help://MS.VSCC.v80/MS.MSDN.v80/MS.NETDEVFX.v20.it/cpref2/html/M_System_Double_IsNaN_1_16514d3a.htm) per determinare se un valore è non numerico.

Nota: Non è possibile determinare se un valore è non numerico confrontandolo con un altro valore uguale a **NaN**.

**Double.NegativeInfinity**

Rappresenta l'infinito negativo. Il valore di questa costante è il risultato della divisione di un numero negativo per zero.

La costante viene restituita quando il risultato di un'operazione è minore di MinValue.

Utilizzare IsNegativeInfinity per determinare se un valore restituisce l'infinito negativo. Non è possibile determinare se un valore restituisce un infinito negativo confrontandolo con un altro valore uguale a **NegativeInfinity**.

**Double.PositiveInfinity**

Rappresenta l'infinito positivo. Il valore di questa costante è il risultato della divisione di un numero positivo per zero.

La costante viene restituita quando il risultato di un'operazione è maggiore di MaxValue.

Utilizzare IsPositiveInfinity per determinare se un valore restituisce l'infinito positivo. Non è possibile determinare se un valore restituisce l'infinito positivo confrontandolo con un altro valore uguale a **PositiveInfinity**.

COSTRUTI DECISIONALI

* **if-else**

L'istruzione **if** controlla che un valore,un carattere,una digitazione corrisponda ad un valore effettivo preimpostato, si usa else se il valore digitato dall utente non corrisponde con l if iniziale. Es.

If(scelta ==1)

{

Console.writeline(“Scelta 1”);

}

Else

{

Console.writeline(“scelta 2”);

}

* Switch

L’insruzione switch va aricercare in un elenco reimpostato il valore digitato dall’utente

Switch(scelta)

{

Case 1:

console.writeline(“Caso1”);

}

Se dopo un case non inserisco un break/continue/goto il ciclo termina

OPERATORI

|  |  |  |
| --- | --- | --- |
| = | | Assegna un valore. |
| = = | | Esegue il test per verificare l'uguaglianza. |
| \*, /, % | | Moltiplicazione, divisione, modulo |
| +, - | | Addizione, sottrazione |
| & | | AND logico |
| ^ | | XOR logico |
| | | | OR logico |
| V++ | Incrementare la variabile v di un'unità. | |
| V += N | Incrementare la variabile v di n unità. | |
| V \*= N | Moltiplicare la variabile v per n unità. | |
| V -= N | Sottrarre n unità dalla variabile v. | |

|  |  |
| --- | --- |
| = = | Esegue il controllo per verificare l'uguaglianza. |
| != | Esegue il controllo per verificare la disuguaglianza. |
| > | Maggiore di |
| < | Minore di |
| >= | Maggiore o uguale a |
| <= | Minore o uguale a |
| && | AND condizionale. Equivale ad e |
| || | OR condizionale. Equivale ad oppure |
| ! | NOT condizionale. Nega |

++x e x++

L'operatore di incremento (++) incrementa il proprio operando di 1 e può essere visualizzato prima o dopo tale operando

Il primo formato rappresenta un'operazione di incremento prefissa. Il risultato dell'operazione corrisponde al valore dell'operando dopo l'incremento.

Il secondo formato rappresenta un'operazione di incremento suffissa. Il risultato dell'operazione corrisponde al valore dell'operando prima dell'incremento.

--x e x--

L'operatore di decremento (**--**) decrementa il proprio operando di 1 e può essere visualizzato prima o dopo tale operando

Il primo formato rappresenta un'operazione di decremento prefissa. Il risultato dell'operazione corrisponde al valore dell'operando dopo il decremento. Il secondo formato rappresenta un'operazione di decremento suffissa. Il risultato dell'operazione corrisponde al valore dell'operando prima del decremento.

CAST E CONVERSIONE TIPI

**Conversioni implicite**: non è richiesta alcuna sintassi speciale perché la conversione è indipendente dai tipi e nessun dato viene perso. Gli esempi includono le conversioni da tipi integrali più piccoli a più grandi. Per i tipi numerici, una conversione implicita può essere eseguita quando il valore da archiviare può essere adattato alla variabile senza essere troncato o arrotondato. Ad esempio, una variabile di tipo long (intero a 8 byte) è in grado di archiviare qualsiasi valore archiviabile in un tipo int

**Conversioni esplicite (cast)**: le conversioni esplicite richiedono un operatore di cast. Il cast è obbligatoria quando le informazioni potrebbero andare perdute durante la conversione o quando la conversione potrebbe non riuscire per altri motivi. Gli esempi tipici includono la conversione numerica in un tipo con meno precisione o un intervallo più piccolo. Se una conversione non può essere eseguita senza un rischio di perdita di informazioni, il compilatore richiede che si esegua una conversione esplicita, chiamata cast. Il cast è un modo di informare in modo esplicito il compilatore che si intende eseguire la conversione e che si è consapevoli che potrebbe verificarsi una perdita di dati. Per eseguire un cast, specificare il tipo di destinazione del cast in parentesi davanti al valore o alla variabile da convertire. Il seguente programma esegue il cast di double su int. Se il cast non è presente, il programma non verrà compilato.

double x = 1234.7;

int a;

// Cast double to int.

a = (int)x;

System.Console.WriteLine(a);

**Conversioni con le classi di supporto**: per eseguire la conversione tra tipi non compatibili,ad esempio tra int e double o tra char e bool ecc. vengono usati i metodi con il Convert con il tryparse o il parse

PARSE: Converte il valore della stringa specificata nell'equivalente carattere Unicode. Es.

Int .parse(“a”);

TRY PARSE:Converte il valore della stringa specificata nell'equivalente carattere Unicode.Un codice restituito indica se la conversione ha avuto esito positivo o negativo. Il metodo TryParse è come il metodo [Parse](http://msdn.microsoft.com/it-it/library/system.char.parse(v=vs.110).aspx), ma il metodo TryParse non genera un'eccezione se la conversione non riesce. Es. result = char.tryparse(“a”, out ,out charval); show(result,”char”,charval.tostring());

VARIABILI E COSTANTI

**Variabili**

Una variabile rappresenta un valore numerico o stringa oppure un oggetto di una classe. Il valore archiviato nella variabile può variare, tuttavia il nome rimane identico. Una variabile corrisponde a un tipo di campo.

**Costanti**

Una costante corrisponde a un altro tipo di campo. Contiene un valore che viene assegnato durante la compilazione del programma e non viene ulteriormente modificato. Per la dichiarazione delle costanti viene utilizzata la parola chiave const o double. Le costanti risultano utili per agevolare la leggibilità del codice.

GENERAZIONE NUMERI CASUALI (variabile mioRand)

// Senza Seed

Random mioRand = new Random();

// Genera un valore random intero.

Console.WriteLine( " Valore generato {0} ", mioRand.Next());

// Genera un valore random intero non negativo inferiore al massimo specificato.

Console.WriteLine( " Valore generato {0} ", mioRand.Next(50));

// Genera un valore random intero compreso tra 50 <= Valore < 100.

Console.WriteLine( " Valore generato {0} ", mioRand.Next(50, 100));

// Genera un valore random double >= 0,0 e < 1,0.

Console.WriteLine( " Valore generato {0} ", mioRand.NextDouble());

|  |
| --- |
| // Con Seed  Random mioRandFisso = new Random(18);  // Genera un valore random intero.  Console.WriteLine(" Valore generato {0} ", mioRandFisso.Next());  // Genera un valore random intero non negativo inferiore al massimo specificato.  Console.WriteLine(" Valore generato {0} ", mioRandFisso.Next(50));  // Genera un valore random intero compreso tra 50 <= Valore < 100.  Console.WriteLine(" Valore generato {0} ", mioRandFisso.Next(50, 100));  // Genera un valore random double >= 0,0 e < 1,0.  Console.WriteLine(" Valore generato {0} ", mioRandFisso.NextDouble()); |

Console.WriteLine();