Tecniche di Programmazione avanzata

Corso di Laurea Specialistica in Ingegneria Telematica
Università Kore – Enna – A.A. 2009-2010

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

Il linguaggio C#
Operatori, Espressioni
Scope, Istruzioni

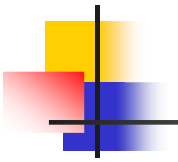
C#: Operatori ed Espressioni

- C# ammette le tipiche espressioni Java:
  - espressioni condizionali
  - operatore + per concatenare stringhe
  - assegnamento multiplo
  - operatori di assegnamento "compatti"
- con le seguenti differenze:
  - gli operatori AND e OR esistono in doppia versione
    - senza (&, |) regola di cortocircuito
    - con (&&, ||) regola di cortocircuito
- NB: le conversioni di tipo sono automatiche se avvengono per promozione, mentre richiedono cast se avvengono per riduzione
- Tutti i tipi inferiori a int sono sempre promossi a int
- I valori decimal sono incompatibili con float e double
- Il booleano è un tipo a se stante (no conversioni)

Operatori

Primary	(x) x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked
Unary	+ - ~ ! ++x --x (T)x
Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational	< > <= >= is as
Equality	== !=
Logical AND	&
Logical XOR	^
Logical OR	|
Conditional AND	&&
Conditional OR	||
Conditional	c?x:y
Assignment	= += -= *= /= %= <<= >>= &= ^= |=

- La priorità va da sinistra a destra
- Gli operatori unari +, -, ~, ! E quelli di cast sono valutati da destra verso sinistra: "- (int) x" equivale a "- ((int) x)"



Espressioni Aritmetiche

Il tipo degli operatori deve essere

- numerico or *char*
- Gli operandi di ++ e -- devono essere numerici o enumerativi (++ e - - si applicano anche a *float* e *double*!)

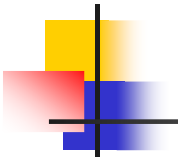
Tipo del risultato

Il più piccolo dei tipi numerici degli operandi (minimo int)

Note

- uint => long
- ulong => illegale

uint • (sbyte | short | int) => long
ulong • (sbyte | short | int | long) => illegale
decimal • (float | double) => illegale



Confronti

Tipo degli operandi

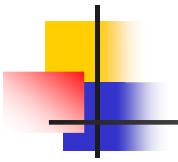
- <, >, <=, >=: numerici, *char*, *enum*
- ==, !=: numerici, *char*, *enum*, *bool*, riferimento
- x is T: *x*: espressione di un qualsiasi tipo
  T: riferimento

esempio:

```
obj is Rectangle
objOfValueType is IComparable
3 is object
arr is int[]
```

Risultato

bool



Espressioni booleane (&&, ||, !)

Tipo degli operandi

bool

Tipo del risultato

bool

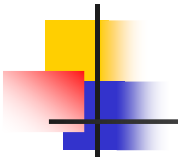
Uso del corto-circuito (short-circuit):

a && b => if (a) b else false

a || b => if (a) true else b

In altre parole:

- a && b: b is evaluated only if a is true
- a || b : b is evaluated only if a is false



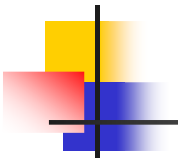
Espressioni con i Bit (&, |, ^, ~)

Tipo degli operandi

- & | ^ : integer type, *char*, *enum*, *bool*
- ~ : integer type, *char*, *enum*
  - The ~ operator performs a bitwise complement operation on its operand.
- Se i tipi degli operandi sono diversi, quello di dimensione minore viene convertito

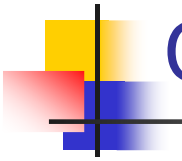
Tipo del risultato

- Tipo di dimensione maggiore
- Per i tipi numerici e *char* il risultato è almeno *int*



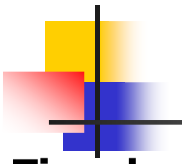
Operatori booleani/bit

- the single "|" and "&" are bit-wise operators, whereas && and || are for logical expressions, so they are used for completely different things.
- using a single & or | evaluates the right hand operand no matter what the left hand one was.
- If you use && or || the right hand operand isn't evaluated if the left invalidates the statement
- Using | and & so that both sides get evaluated is not the best choice; write better code that doesn't required both sides to evaluate instead.



Operatori booleani/bit

```
using System;
class Operators {
    static void Main(){
    string s=String.Format("0101 | 1010 fa : {0:X}", (0x5 |0xA));Console.WriteLine(s)
    s=String.Format("0101 & 1010 fa : {0:X}", (0x5 & 0xA));    Console.WriteLine(s);
    s=String.Format("1000 & 1010 fa : {0:X}", (0x8 & 0xA));    Console.WriteLine(s);
    s=String.Format("1000 | 1010 fa : {0:X}", (0x8 | 0xA));    Console.WriteLine(s);
    int x=0;
    /* LA SEGUENTE COSTRUZIONE NON E' VALIDA...
    if (0x5 & 0xA) Console.WriteLine("Il risultato di (0x5 & 0xA) ha dato true");
    else Console.WriteLine("Il risultato di (0x5 & 0xA) ha dato false");
    MESSAGGIO DEL COMPILATORE: operators.csc(17,7): error CS0031: Impossibile
    convertire il valore costante '0' in 'bool'.
    */
    if (x==(0x5 & 0xA)) Console.WriteLine("Il risultato di (0x5 & 0xA) ha dato true");
    else Console.WriteLine("Il risultato di (0x5 & 0xA) ha dato false");
    }}
```

Shift

Tipo degli operandi per x << y e x >> y

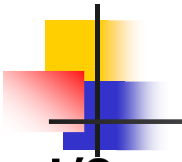
- *x*: intero o *char*
- *y*: *int*

Tipo del risultato

Tipo di *x*, (almeno int)

Note

- esegue uno shift **logico** (high-order bits are zero filled) per gli unsigned types
- esegue uno **shift aritmetico** (high-order empty bits are set to the sign bit) per i signed
- int k=-12;
- uint ku=+12;
- Console.WriteLine("shifting int..."+(k>>1)); // stampa -6 (non sposta il segno)
- Console.WriteLine("shifting uint..."+(ku>>1)); // stampa 6



Overflow

L'Overflow non è verificato automaticamente…

```
int x = 1000000;
x = x * x;   // -727379968, nessun errore !
```

Ma può essere attivato:

```
x = checked(x * x);  // → System.OverflowException

checked {
 …
 x = x * x;        // → System.OverflowException
 …
}
```

è possibile attivare il check con una opzione del compilatore
csc /checked Test.cs

typeof e sizeof

typeof

- Restituisce il descrittore del tipo di un oggetto (*o.GetType()*).

```
Type t = typeof(int);
Console.WriteLine(t.Name);     // → Int32
```

sizeof

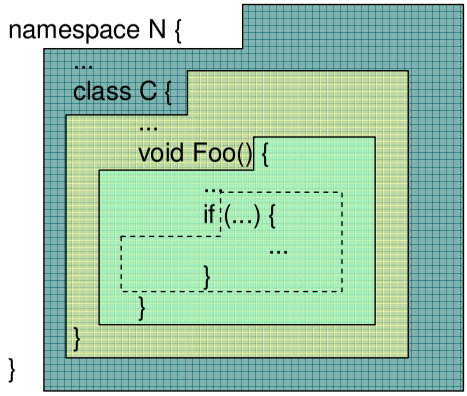
- Restituisce le dimensioni del tipo in byte.
- Può essere applicato solo ai tipi valore
- Può essere usato solo in blocchi unsafe:

```
unsafe {
    Console.WriteLine(sizeof(int));
    Console.WriteLine(sizeof(MyEnumType));
    Console.WriteLine(sizeof(MyStructType)); }
```

- The **unsafe** keyword denotes an unsafe context, which is required for any operation involving pointers. To compile unsafe code, you must specify the /unsafe compiler option. Unsafe code is not verifiable by the common language runtime.

Ambiente e visibilità

- **namespace**:
  - classi, interfacce, structs, enums, delegates
- **class, interface, struct**:
  - fields, metodi
- **enumeration**:
  - Costanti enumerative
- **block**:
  - Variabili locali



I blocchi di istruzione permettono la creazione di variabili locali, ma non definiscono l'ambiente o la visibilità delle variabile

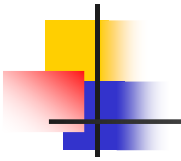
Scope

- Un nome non può essere dichiarato più di una volta nello stesso ambiente e livello, può però essere dichiarato in ambiente interno (eccetto nei blocchi di istruzione), ad esempio:

```
using System;
class Operators {
   static void Main(){
        int x=6;
        if (3>2) {          int x=4;       }    // NON PERMESSO
        for (int f=0;f<3;f++){      int x=5;       }    // NON PERMESSO
   }
   void metodo1() {       int x=12;          }    // PERMESSO
   }
```

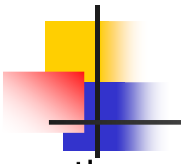
Il compilatore risponde con:

- operators.csc(5,7): error CS0136: Una variabile locale denominata 'x' non può essere dichiarata in quest'ambito perché darebbe un significato diverso a 'x', che è già utilizzato in un ambito 'padre o corrente' per identificare qualcos'altro
- operators.csc(6,7): error CS0136: Una variabile locale denominata 'x' non può essere dichiarata in quest'ambito perché darebbe un significato diverso a 'x', che è già utilizzato in un ambito 'padre o corrente' per identificare qualcos'altro



Visibilità

- Un nome (namespace, classe, struct, interface) è visibile nell'intero ambiente.
- Un nome può essere usato prima della sua dichiarazione (eccetto che per le variabili locali)
- Se un nome è dichiarato in un ambiente più interno nasconde le precedenti dichiarazioni
- Nessun nome è visibile fuori dall'ambiente di dichiarazione
- La visibilità di un nome è controllata dai modificatori *public*, *private*, *protected* and *internal*.
- I nomi delle costanti enumerative sono accessibili se qualificate dal nome dell'enumerazione.



Namespace

- the reasons for introducing namespaces into the C# language is to avoid name collisions when using libraries from multiple vendors
- In addition, you are free to create your own. You do this by using the namespace keyword, followed by the name you wish to create.

```
namespace A {
      ... classes ...
      ... interfaces ...
      ... structs ...
      ... enumerations ...
      ... delegates ...
    namespace B {  // full name: A.B
          ...
    }
}
```

```
namespace A {
    ...
    namespace B {...}
}
```

```
namespace C {...}
```

Uso di più Namespace

Color.cs

```
namespace Util {
  public enum Color
{...}
}
```

Figures.cs

```
namespace Util.Figures {
  public class Rect {...}
  public class Circle {...}
}
```

Triangle.cs

```
namespace Util.Figures {
  public class Triangle {...}
}
```

```
using Util.Figures;
class Test {
      Rect r;   // non necessita di qualificatore)
      Triangle t;
      Util.Color c;  // qualificatore
}
```

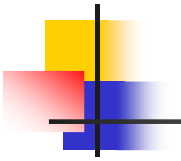
Foreign namespace

- Devono essere importati (e.g. *using Util;)*
- O qualificati (e.g. *Util.Color*)

Visibilità in Classi e Interfacce

```
class C {  // applies also to structs
    ... fields, constants ...
    ... methods ...
    ... constructors, destructors ...
    ... properties ...
    ... indexers ...
    ... events ...
    ... overloaded operators ...
    ... nested types (classes, interfaces, structs, enumerations, delegates)
    ...
}
interface IX {
    ... methods ...
    ... properties ...
    ... indexers ...
    ... events ...
}
```

Lo spazio di una sottoclasse non appartiene allo spazio del genitore
=> È possibile usare gli stessi nomi nella classe e nella sottoclasse

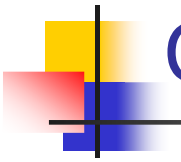


Blocchi

Tipo dei blocchi di istruzioni

```
void foo (int x) {     // method block
    ... local variables ...
    if (...) {       // nested block
        ... local variables ...
    }
    for (int i = 0; ...) {  // nested block
        ... local variables ...
    }
}
```

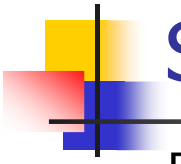
- L'ambiente dei blocchi interni coincide con quello esterno
- I parametri formali appartengono all'ambiente del metodo
- La variabile dei cicli appartiene al blocco del ciclo



C#: Istruzioni

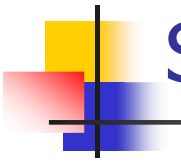
- C# ha le tipiche istruzioni C, C++, Java, con le seguenti differenze:
  - introduce 4 nuove istruzioni: *foreach*, *checked*, *lock* and *using*
  - il blocco, pur costituendo uno scope per la definizione di variabili locali, non consente di definire variabili *omonime* ad altre definite a un livello più esterno *(non consente di "nascondere" nomi più esterni)*
  - l'istruzione switch richiede obbligatoriamente la presenza di break dopo ogni case non vuoto

```
switch(espressione) {
case 0: istruzione; break;
case 1:
case 2: istruzione; break;
}
```



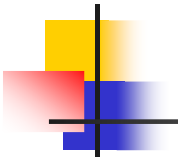
Statement lists

- Block statements
- Labeled statements
- Declarations
  - Constants
  - Variables
- Expression statements
  - checked, unchecked
  - lock
  - Using
- Conditionals
  - if
  - switch
- Loop Statements
  - while
  - do
  - for
  - foreach
- Jump Statements
  - break
  - continue
  - goto
  - return
  - throw
- Exception handling
  - try
  - throw



Sintassi istruzioni

- Le istruzioni terminano con punto e virgola (;)
  - Come C, C++ e Java
- L'istruzione { ... } non ha bisogno del punto e virgola
- Commenti
  - // commenti di una linea, come C++
  - /* commenti di piu linee, come C */

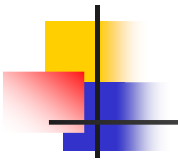


Istruzioni elementari

- Istruzione vuota
  ; // il ; fa parte dell'istruzione

- Assegnazione
  x = 3 * y + 1;

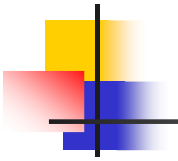
- Invocazione di un metodo
  string s = "a,b,c";
  string[] parts = s.Split(','); // oggetto
  s = String.Join(" + ", parts);// metodo di classe (statico)



Istruzione if

- Identica al C, C++, Java:

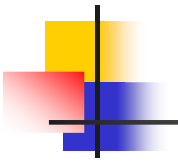
```
if ('0' <= ch && ch <= '9')
  val = ch - '0';
else if ('A' <= ch && ch <= 'Z')
  val = 10 + ch - 'A';
else {
  val = 0;
  Console.WriteLine("invalid character " + ch);
}
```



switch Statement

```
switch (country) {
    case "England": case "USA":
     language = "English";
     break;
    case "Germany": case "Austria": case "Switzerland":
     language = "German";
     break;
    case null:
     Console.WriteLine("no country specified");
     break;
    default:
     Console.WriteLine("don't know the language of " + country);
     break;
}
```

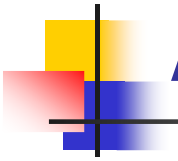
- Tipo dell'espressione intero, char, enum or string (null come etichetta).
- Ogni istruzione deve essere chiusa da break (o return, goto, throw), a differenza di C e Java



switch e Goto

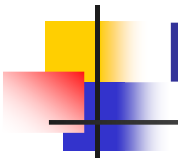
```
int state = 0;
int ch = Console.Read();
switch (state) {
   case 0:  if (ch == 'a') { ch = Console.Read(); goto case 1; }
            else if (ch == 'c') goto case 2;
            else goto default;
   case 1:  if (ch == 'b') { ch = Console.Read(); goto case 1; }
            else if (ch == 'c') goto case 2;
            else goto default;
   case 2:  Console.WriteLine("input valid");
            break;
   default: Console.WriteLine("illegal character " + ch);
            break;
}
```

- Permette l'implementazione delle FSM



foreach

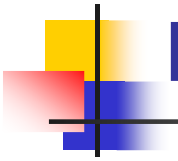
- Oltre ai classici while, do, for, per i cicli esiste l'istruzione foreach, che itera su una collezione di oggetti
- Un tipo C è considerato una collezione se implementa System.IEnumerable o soddisfa leseguenti condizioni:
  - C contiene una istanza pubblica di un metodo con signature GetEnumerator() che ritorna E che è un *struct-type*, *class-type*, o *interface-type*.
  - E contiene un istanza pubblica di un metodo con signature MoveNext() e che ritorna bool.
  - E contiene un istanza pubblica di una proprietà chiamata Current che permette di leggere il valore corrente. Il tipo di questa proprietà è detto essere il tipo degli elementi della collezione



Esempio:foreach

```
object[] a = new int[]{ 1, 2, 3, 4 };
foreach (int i in a) {
  Console.WriteLine(i);
}
```

- L'istruzione alloca una variabile locale chiamata i e usa il metodo enumeration per iterare sopra la collezione assegnando alla variabile il valore corrente
- Un cast implicito viene eseguito per il tipo della variabile utilizzato come variabile locale

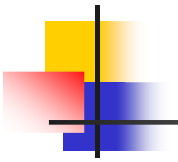


Esempio:foreach

```
int[] a = {3, 17, 4, 8, 2, 29};
foreach (int x in a) sum += x;

string s = "Hello";
foreach (char ch in s) Console.WriteLine(ch);

Queue q = new Queue(); // gli elementi sono
object
q.Enqueue("Pippo"); q.Enqueue("Pluto"); ...
foreach (string s in q) Console.WriteLine(s);
```

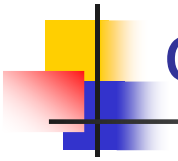
Salti incondizionati

break; (non esiste il break con etichetta di Java)

continue;

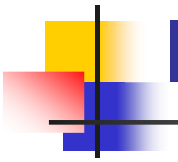
goto case 3:

myLab:
...
goto myLab;



checked/unchecked

- Con le keywords checked and unchecked il programma può controllare il comportamento delle operazioni aritmetiche
- La sintassi è la seguente:
  - checked(expr)
  - unchecked(expr)
- Le keywords possono essere utilizzate anche come istruzioni per marcare blocchi di istruzioni: checked {…}

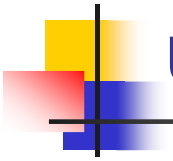


lock

- lock(x) è un abbreviazione per:

```
System.Threading.Monitor.Enter(x);
try {
 ...
} finally {
  System.Threading.Monitor.Exit(x);
}
```

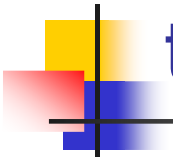
- x deve essere un'espressione con reference type



using

- using(R r1 = new R()) { r1.F(); }
- È equivalente a::

```
R r1 = new R();
try {
  r1.F();
} finally {
 if (r1 != null) ((IDisposable)r1).Dispose(); }
```

typeof, is, as

- Sono operatori per lavorare con i tipi:
  - typeof ritorna il tipo dell'oggetto di una data class:

    Type t = typeof(string);
  - is è un operatore booleano che esegue il test se è corretto eseguire il case di un riferimento con un dato tipo:

    if (t is object) { … }
  - as permette di eseguire un cast che potrebbe fallire

```
string s = o as string;
if (s == null)
  Console.WriteLine("Not a string");
```