TECNICO SUPERIORE WEB DEVELOPER FULL STACK

ITS DIGITAL ACADEMY
MARIO VOLPATO

niuk
innovation and knowledge

TECNICO SUPERIORE WEB DEVELOPER FULL STACK

#1 - Presentazione e introduzione al corso

ITS DIGITAL ACADEMY
MARIO VOLPATO

niuko
innovation and knowledge

Presentiamoci!

Michele Caceffo

michele.caceffo@itsdigitalacademy.com

Potete scrivermi, prima o poi vi rispondo.

Chi sono?

- **Sono un informatico**: LM in Informatica presso l'Università di Trento;
- **Sono in parte un biologo**: specializzazione in Bio-Informatica;
- **Sono un insegnante**: Informatica, Sistemi e Reti, TPSIT presso l'ITT "Chilesotti" di Thiene;
- **Sono stato insegnante del corso di Algoritmi** presso l'ITS Kennedy di Thiene
- **Sono (stato) docente-assistente** al corso universitario di Linguaggi Formali e Compilatori.

E voi?

Panoramica del corso

Innanzitutto

Il corso si appoggia a Classroom, quindi materiali, consegne e il test di valutazione finale si troveranno tutti lì.

Codice del corso

pxr4uif

Prerequisiti

- Variabili e tipi di dato semplici
- Istruzioni di controllo
  - Sequenzialità dei programmi
  - Selezione
  - Ripetizione definita e indefinita
- Funzioni e passaggio di parametri (!)
- Il concetto di Array

Rivedremo questi concetti tra poco, giusto per fissare un vocabolario comune e non confonderci.

Cosa è previsto che facciate? Conoscenze

- Principi generali della programmazione nei linguaggi moderni;
- Differenze concettuali tra variabili e operatori;
- Implicazioni relative alla complessita' computazionale;
- Conoscere i principali algoritmi di ordinamento: Bubble sort, Merge sort, Quick sort;
- Concetti e implicazioni relative alle strutture dati: insiemi, dizionari e tabelle di hash.

Cosa è previsto che facciate? Abilità

- Utilizzare l'ambiente di sviluppo e il debug;
- Utilizzare i flussi di controllo: la selezione e i cicli;
- Utilizzare stringhe e puntatori
- Scrivere codice per la realizzazione di algoritmi di vista per array, ricerca lineare, ricerca binaria per array ordinati;
- Gestire liste, code, stack e algoritmi di vista, ricerca, inserimento e rimozione;
- Utilizzare correttamente alberi binari di ricerca e algoritmi di vista, ricerca, inserimento e rimozione;
- Gestire l'accesso ai file.
- Utilizzare librerie audio e bitmap

Programmerete?

Mi spiace dirlo, ma temo poco 😔, per due motivi:

1. Gli algoritmi “belli” che andremo ad analizzare sono già implementati da tutti i linguaggi di programmazione seri;
2. Molto del lavoro dell’analista di algoritmi si fa con carta, penna e testa.

Detto questo, quando affronteremo esempi di algoritmi “belli” e “non così belli” siete invitati a provare ad implementarli, in modo da assimilarne il funzionamento.

Stesso discorso per le Strutture Dati: molto di quello che vedrete è già implementato nelle varie librerie. Però “molto” non è “tutto”... 😉

"An algorithm must be seen to be believed, and the best way to learn what an algorithm is all about is to try it"

Donald Knuth,
“The art of computer programming”

In questo corso parleremo di:

Algoritmi

- Concetti di base di programmazione (Ripasso)
- Analisi algoritmi
  - Notazione O, Ω, Θ
  - Problemi VS algoritmi
- Ricorsione e algoritmi di ricerca

Tecniche di programmazione

- Divide-et-impera
- Backtracking

Strutture dati

- Liste
- Insiemi e Dizionari
- Hashing e Hashset
- Alberi e Grafi
  - Algoritmi di visita
  - BFS e DFS

Se avanziamo tempo

- Algoritmi greedy
- Teoria dell'NP-completezza
- Problemi intrattabili

Regole di convivenza civile

1. Fate domande!
   - Se sono poco chiaro, chiedete ulteriori spiegazioni;
   - Se volete ulteriori approfondimenti, chiedete: non conosco tutte le risposte – ma so dove cercarle!
2. Date risposte!
   - 4 ore in cui parlo solo io non è una lezione, è un suicidio di massa: voi morite di noia, io muoio per disseccamento delle corde vocali.

"Domanda e sembrerai sciocco per un minuto,
non domandare e resterai sciocco per sempre."

Convenzioni di colore - 1

Quando una slide ha la matita, di solito sono esercizi (e di solito lo scrivo in alto, nel titolo).

Se la matita è VERDE, sono esercizi guidati, cioè li faccio io per mostrarvi un esempio o due. Poi vi giro anche il codice che ho scritto, ma sono comunque altri spunti per esercitarvi.

Quindi fateli.

Convenzioni di colore - 2

Se la matita è ARANCIONE, sono esercizi da fare in classe, ossia vi do il tempo di farli e poi li correggiamo insieme; io preparo una soluzione, ma mi piacerebbe vedere le vostre, prima.

Quindi fateli.

Convenzioni di colore - 3

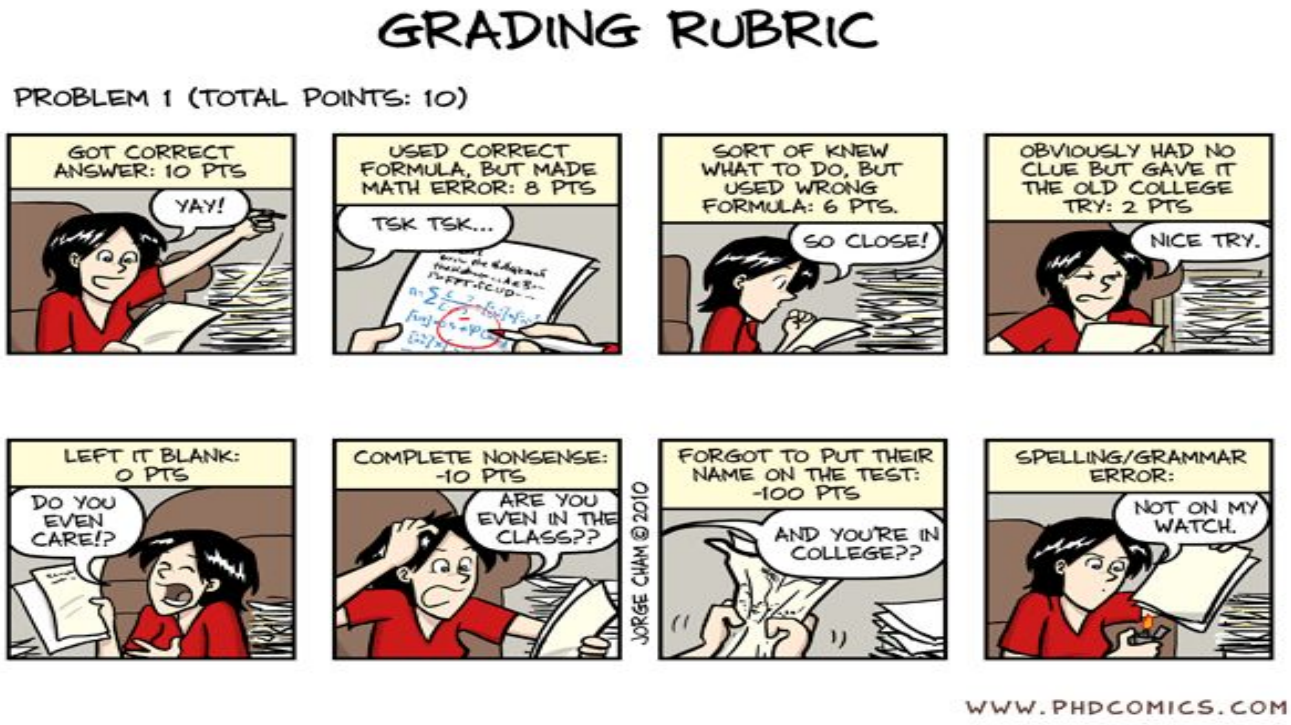
Se la matita è ROSSA, sono esercizi da fare per casa, nel senso che li lascio come esercitazione e di solito li correggo la lezione successiva, io o - più probabilmente - voi.

Quindi fateli.

Valutazioni - 1

La valutazione unica (con i relativi recuperi, se necessario), effettuata tramite un quiz su classroom.

Valutazioni - 2

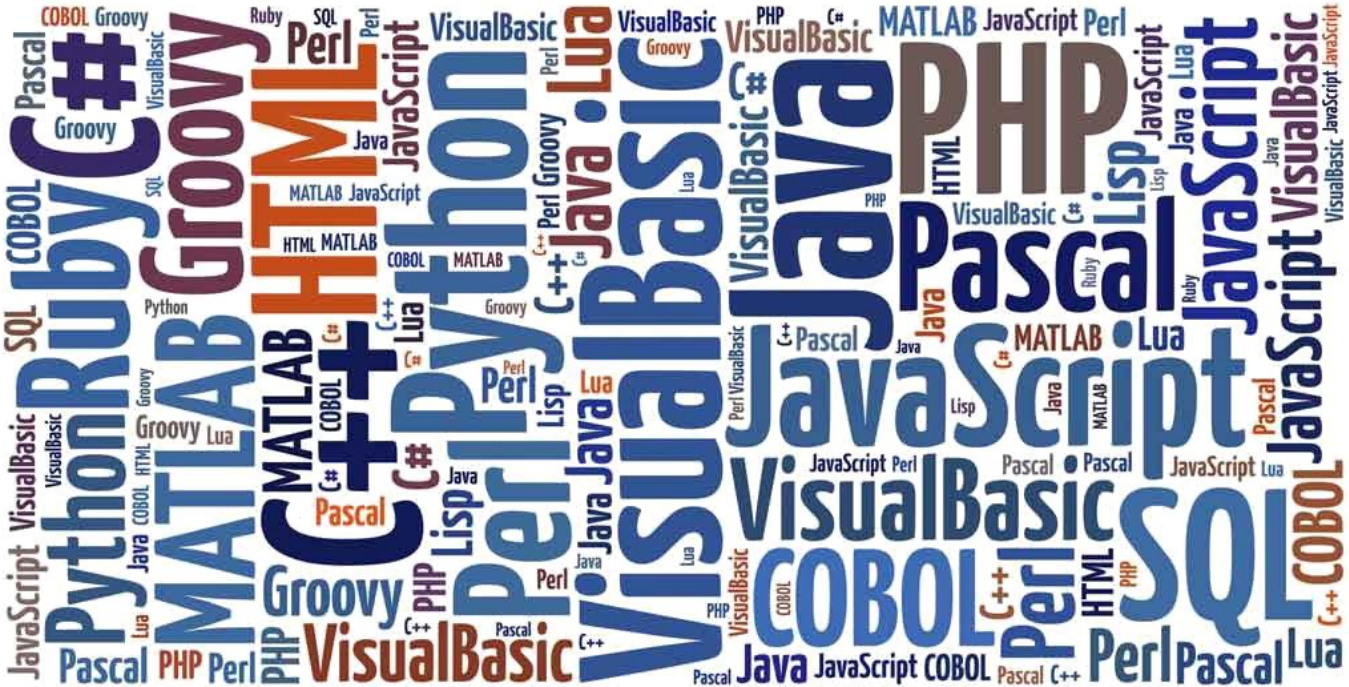


Cominciamo!

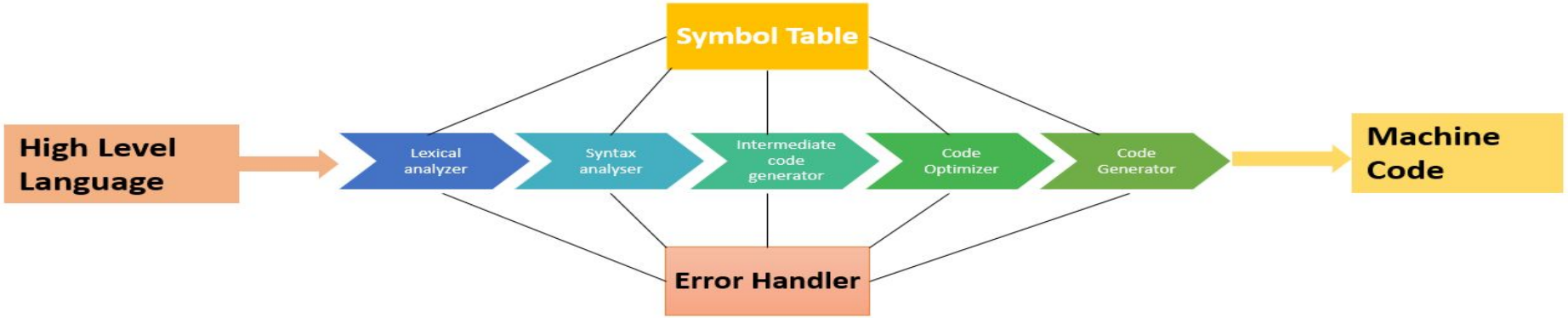
Che cos’è un linguaggio di programmazione?

“Un linguaggio artificiale usato per scrivere **istruzioni** che possono essere tradotte in **linguaggio macchina** ed essere poi eseguite da un computer.”

“Codice di **parole** e **simboli** riservati usati nei programmi per computer, che danno istruzioni al computer su come realizzare certi compiti.”


COBOL Groovy Ruby SQL Perl VisualBasic PHP C# MATLAB JavaScript Perl
Pascal C# Groovy HTML Python Java Lua
VisualBasic Groovy JavaScript Perl Lisp
Ruby SQL Perl Java C++ Pascal
Python Groovy Lua Ruby
MATLAB HTML C++ Lua
JavaScript Lisp Lua
Groovy Perl Pascal
VisualBasic PHP
Java
COBOL
SQL

Compilazione: le fasi

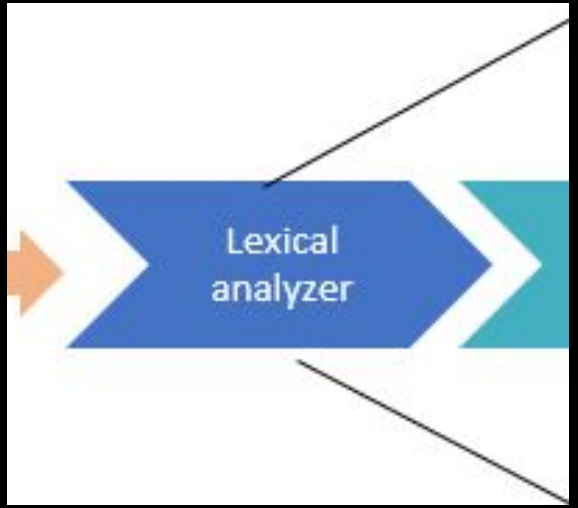


Analizzatore lessicale

Esamina il codice sorgente alla ricerca di parole chiave, identificando ad es.:

- Simboli (+, -, {, …)
- Valori (3.14, 5, k)
- Identificatori (a, pippo, dividendo, tasso_interesse…)

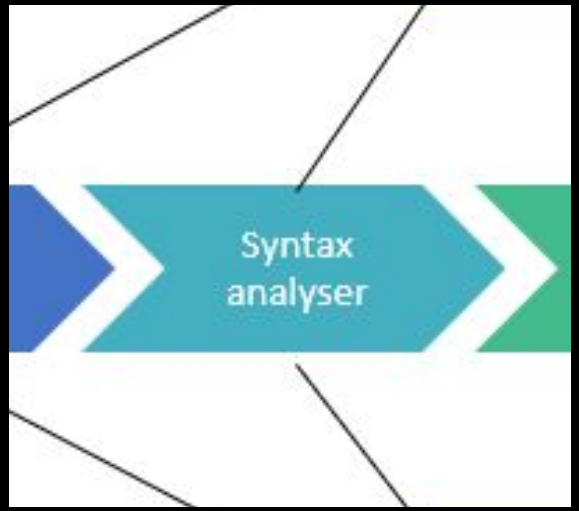
Al termine, produce il c.d. Token Set



Analizzatore sintattico

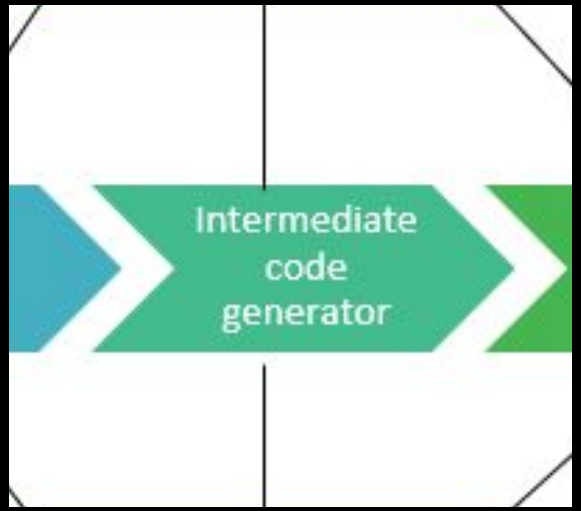
Esamina il Parse Tree e produce l'Albero Sintattico, dove vengono riconosciute:

- Precedenze degli operatori (la somma dopo la moltiplicazione, ecc.)
- Chiamate a funzione, dichiarazioni, comandi, espressioni



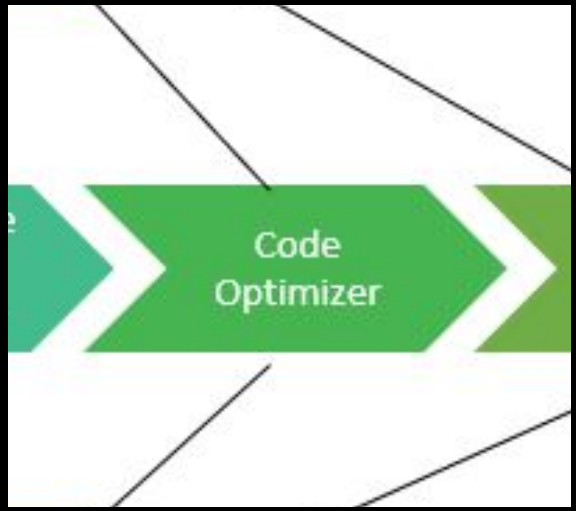
Generatore di codice intermedio

Trasforma l'Albero Sintattico in una rappresentazione intermedia del programma sorgente, non dipendente dalla macchina, ma simile alle istruzioni macchina.



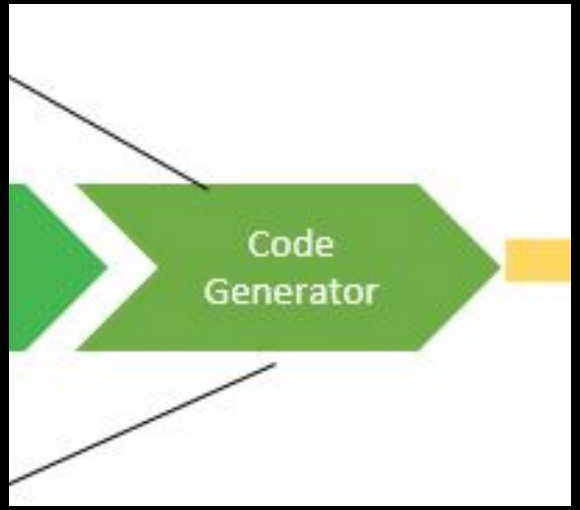
Ottimizzatore del codice

Prende l'output della fase precedente - c.d *Codice Oggetto* o *Codice Binario* - e prova a sostituire sezioni identificabili del codice con altre istruzioni algoritmicamente più efficienti.



Generatore del codice (macchina)

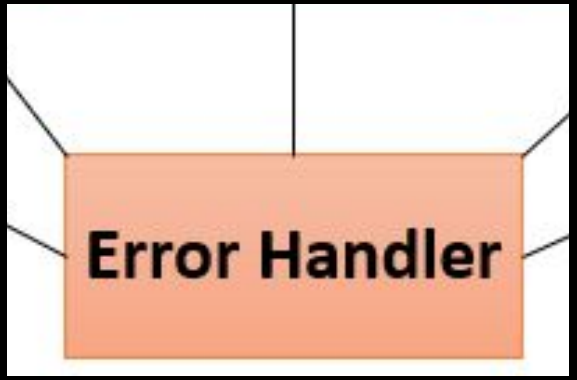
Dopo l'ottimizzazione, al codice oggetto vengono aggiunte le librerie e altri elementi (fase di *linking*), quindi viene tradotto in istruzioni per il processore.



Gestione degli errori

Gli errori possono comparire durante tutte le fasi della compilazione, ma i principali sono:

- Errori sintattici
- Errori semantici



Errori sintattici VS Errori semantici

Gli errori sintattici riguardano il “come abbiamo scritto”:

- Variabili non dichiarate
- Variabili che cambiano nome
- Sintassi sbagliata dei comandi

Gli errori semantici riguardano “cosa vogliamo dire”:

- Il cane mangia il panino
- Il panino mangia il cane

Paradigmi dei linguaggi di programmazione

I linguaggi di programmazione possono essere classificati in base al paradigma, ossia il modello utilizzato dal linguaggio per descrivere i programmi:

- **Imperativo (o procedurale)**
- **Ad oggetti**
- Funzionale
- Dichiarativo
- Concorrente

Introduciamo C# e Visual Studio

C# (C-sharp) è un linguaggio di programmazione ad oggetti nato nel 2001 da Microsoft.

E' nato insieme al framework .NET (dot-net), un insieme di funzionalità, librerie, costrutti e componenti di vario genere utilizzabili per scrivere programmi con molteplici scopi.

Visual Studio (oggi alla versione 2022) è l'ambiente di sviluppo (IDE) principale per creare applicazioni usando C# e il framework .NET.

La versione community è gratuita, con alcune limitazioni sulle funzionalità che non ci interessano.

Tipi di dato

Quando scriviamo un programma, dobbiamo partire dalla definizione degli elementi che utilizzeremo per modellare il nostro problema, le **variabili**.

Una variabile è *un contenitore per una informazione*.

Ogni variabile è definita da un **nome** e da un **tipo**.

Il tipo è ciò che definisce le caratteristiche della nostra variabile:

- quali valori può assumere
- quali operazioni sono permesse

Tipi di dato predefiniti

- Numerici: int, float, double
- Caratteri: char
- Valori di verità: bool

Operazioni sui tipi numerici

- + somma
- - sottrazione
- * moltiplicazione
- / divisione (attenzione al tipo!)
- % resto della divisione intera
- ++ incremento (prefisso o postfisso)
- -- decremento (prefisso o postfisso)

Operazioni sui tipi booleani

- && AND condizionale
- & AND logico
- || OR condizionale
- | OR logico
- ! NOT condizionale
- ^ XOR logico

Operazioni di confronto

- == uguaglianza
- != diversità
- > maggiore
- >= maggiore o uguale
- < minore
- <= minore o uguale

Le stringhe

Definizione

Una stringa è una sequenza di caratteri.

Una stringa è un oggetto di tipo `String` il cui valore è testo. Internamente, il testo è memorizzato come una collezione sequenziale **read-only** di oggetti `Char`.

In C#, la parola chiave `string` è un alias per `String`. `String` e `string` sono quindi equivalenti, e potete usare la convenzione che preferite.

L’immutabilità delle stringhe

- Gli oggetti stringhe sono *immutabili*: non possono essere modificati dopo che sono stati creati.
- Tutti i metodi e gli operatori C# che *sembrano* modificare una stringa restituiscono effettivamente i risultati in un **nuovo oggetto** stringa.
- Ad esempio, nell’espressione s1 += s2 le due stringhe originali non sono modificate. L'operatore += crea una nuova stringa che contiene i contenuti combinati e che viene assegnata alla variabile s1.

Confronto tra stringhe

Si confrontano le stringhe per rispondere a domande come "Queste due stringhe sono uguali?" o "In quale ordine queste stringhe dovrebbero essere posizionate?”

Fattori che influenzano il confronto delle stringhe:

- Confronto ordinale o linguistico
- Il caso
- Confronti culturali
- Confronti linguistici dipendono dalla `CultureInfo` e dalla piattaforma.

Le modalità per confrontare

- Il metodo String.Compare
- Il metodo Equals
- L'operatore == (e !=)
- Il metodo CompareTo

Il metodo String.Compare(a,b)

Il metodo String.Compare può essere usato per vedere se due stringhe sono uguali o quale stringa è maggiore in termini di valore ASCII. Il metodo Compare restituisce tre possibili valori come int:

- Se il valore restituito è 0, allora entrambe le stringhe sono uguali.
- Se il valore restituito è 1 o maggiore di 0 allora la prima stringa è maggiore della seconda.
- Se restituisce -1 o meno di 0, la seconda stringa è maggiore.

Il metodo a.Equals(b)

Il metodo Equals della stringa viene utilizzato per determinare se due oggetti stringa sono uguali o meno. Il metodo Equals restituisce un booleano:

- True se le stringhe sono uguali
- False altrimenti

Attenzione: il metodo Equals testa l'uguaglianza, non l'identità.

L’operatore a == b

Per questioni che vedremo nel modulo OOP, il metodo Equals e l’operatore == sono strettamente legati e dipendono l’uno dall’altro (insieme al metodo GetHashCode).

Conseguentemente, le due strategie sono equivalenti.

Il metodo a.CompareTo(b)

Confronta questa **istanza** con un oggetto o una stringa specificata e restituisce un intero che indica se questa istanza precede, segue o appare nella stessa posizione nell'ordine dell'oggetto o della stringa specificata.

- Se il valore restituito è 0, allora entrambe le stringhe sono uguali.
- Se il valore restituito è 1 o maggiore di 0 allora la prima stringa è maggiore della seconda.
- Se restituisce -1 o meno di 0, la seconda stringa è maggiore.

In sintesi...

```
string s1 = “Grande Puffo”;
string s2 = “Grande Puffo”;

if (s1 == s2 &&
    s1.Equals(s2) &&
    string.Compare(s1, s2) == 0 &&
    s1.CompareTo(s2) == 0)
    Console.WriteLine(“Le due stringhe sono uguali”);
else
    Console.WriteLine(“Le due stringhe sono diverse”);
```

Questo programma stamperà a video “Le due stringhe sono uguali”.

Esercizi

1. Si realizzi un programma console che calcoli le dimensioni di un foglio in formato AN, con N fornito in ingresso.

   SUGGERIMENTO: Un foglio di carta in formato A0 ha dimensioni 1189 x 841 mm. Un foglio in formato A1 ha il lato lungo uguale al lato corto del formato A0 (841 mm) ed il lato corto uguale alla metà del lato lungo (1189 / 2 mm).

2. Scrivere un programma che legga una sequenza di numeri terminata da 0 e verifichi se sono tutti pari.