Mirella Mussatti

Informatica e programmazione

E-Book di Informatica: algoritmi e introduzione al linguaggio C# per il triennio

Volume 1

Garamond
[didattica digitale]

Via Tevere, 21 Roma

Prima edizione
Volume 1

Cod. ISBN 978-88-86180-92-4

INDICE

1. Informatica: scienza e tecnologia

Si richiamano i principi di base relativi alla disciplina, senza entrare in dettagli, ma come riferimento culturale, in quanto la consapevolezza e la contestualizzazione delle conoscenze sono elementi importanti per il loro rafforzamento: non è invece obiettivo di questo testo fornire gli elementi di base della ICT.

In questa sezione si presentano inoltre gli strumenti che saranno utilizzati per apprendere i principi della programmazione, in particolare .NET Framework 3.5 , gli ambienti di sviluppo Microsoft® Visual Studio® e SharpDeveloper, il linguaggio di programmazione C#. Si esplicitano le scelte effettuate, affinché anche il lettore ne sia consapevole e le possa condividere.

1.1. INFORMATICA: ASPETTI SCIENTIFICI E TECNOLOGICI DELLA DISCIPLINA

PREREQUISITI

Conoscere i concetti fondamentali delle Tecnologie dell'Informazione e della Comunicazione a livello di Patente Europea del Computer[I] (ECDL modulo 1)

OBIETTIVI

Comprendere l'importanza di associare la preparazione tecnica che verrà acquisita nel corso dello studio con una preparazione culturale che permetta di affrontare il rapido evolversi del settore.

Conoscere l'ambiente software che verrà utilizzato per apprendere i principi della programmazione.

1.1.1. Informatica

Alcune riflessioni sulla parola Informatica e sui diversi significati con cui viene utilizzata in contesti differenti.

1.1.1.1. Definizione di informatica

Il termine **informatica** deriva dal francese *information automatique* e si riferisce alla **gestione automatica della informazione**.

Per *informazione*[2] possiamo considerare una qualsiasi forma di notizia o nuova conoscenza; gestione *automatica* significa che le attività possono essere svolte da una macchina, quindi appunto in modo automatico!

In inglese il termine corrispondente è *computer science* che pone l'accento sugli aspetti scientifici e sui fondamenti teorici, come evidenziato dalla affermazione di Edsger Dijkstra, uno dei padri fondatori della Computer Science, il quale sostiene che *Computer science is no more about computers than astronomy is about telescopes* (La scienza dell'informazione ha a che fare con i computer non più di quanto l'astronomia abbia a che fare con i telescopi)[3].

[I] Si è scelto questo riferimento pe rché definito a livello europeo e indipendente dai fornitori .
Per informazioni sulla certificazione ECDL vedi http://www.ecdl.it/ ; per informazioni specifiche circa conoscenze e competenze corrispondenti al primo livello di certificazione, modulo 1 - Concetti di base dell' ICT vedi http://aicanet.net/certificazioni/ecdl/core-level/syllabus; come supporto per l' apprendimento vedi http://www.caspur.it/formazione/mais/html/Home.html

[2] Il concetto rigoroso di informazione, corrispondente a quello fisico di entropia, è stato formulato da Shannon, "A Mathematical Theory of Communication", The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656, July, October, 1948.

[3] Il computer cui si fa riferimento è un sistema che può essere basato su una qualsiasi tecnologia, potrebbe essere uno strumento meccanico, non necessariamente quindi uno strumento digitale.

In Italia si parla di *elaboratore*: il lessico sottolinea gli aspetti applicativi della disciplina. Di qui affermazioni quali: "informatica è la scienza e la tecnica della elaborazione dei dati"[4], oppure "informatica è la scienza applicata che studia le modalità di raccolta, di trattamento e di trasmissione delle informazioni mediante elaboratori elettronici"[5].

Esaminiamo la definizione proposta da ACM (Association for Computing Machinery[6]): **informatica è studio sistematico degli algoritmi** che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione e applicazione.

Risulta centrale il concetto di algoritmo, cioè di sequenza di istruzioni che devono essere eseguite ad esempio da parte di un computer, che come esecutore si rivela essere particolarmente preciso ed efficiente.

Concludiamo ricordando che più in generale con il termine *informatica* ci si riferisce a:

conoscenza ed utilizzo del computer come strumento in sé;

utilizzo del computer quale strumento per lo svolgimento di attività di vario genere.

Ciò deriva, al di là del significato specifico del termine, dall'immaginario collettivo della nostra società con il ruolo assunto in essa dall' "informatica", o meglio dai "sistemi informatici" e naturalmente dall'esperienza quotidiana di coloro che hanno a che fare con il computer; da riflettere da una parte sul processo di sviluppo della società dell'informazione[7], dall'altra parte sulle situazioni di digital divide[8].

1.1.1.2. IT Information Technology

L'espressione **information technology** si riferisce agli aspetti tecnologici del trattamento delle informazioni, in particolare all'insieme di tutte le tecnologie per elaborazione, memorizzazione e utilizzo delle informazioni.

1.1.1.3. ICT Information and Communication Technology

ICT è l'acronimo di **information and comunication technology**: ci si riferisce all'insieme delle tecnologie informatiche (IT) e della telecomunicazione (TLC) che consentono l'eleborazione e lo scambio delle informazioni attraverso le macchine in rete.

Per un maggiore approfondimento vedi il *Museo dell'informatica e del calcolo scientifico* http://www.museoaica.it .

---

[4] Vocabolario della lingua italiana, lo Zingarelli , 2005

[5] Il Sabatini Coletti, Dizionario della Lingua Italiana, Rizzoli LAROUSSE

[6] La Association for Computing Machinery, fondata nel 1947, è nel mondo la principale associazione scientifica ed educativa dedicata al calcolo automatico e all'informatica http://www.acm.org

[7] Portale della Commissione europea sulla società dell' Informazione: http://ec.europa.eu/information_society/index_it.htm

[8] Gap qualitativo e quantitativo nell' uso dell' ICT, in particolare di Internet.

1.1.2. Le professioni dell'ICT

I lavori e le professioni in ambito informatico.

1.1.2.1. Utenti

In ogni ambiente lavorativo, ma anche per attività di svago, si utilizzano i computer quali strumenti di memorizzazione, elaborazione e trasmissione delle informazioni: di qui l'importanza della diffusione delle competenze informatiche di base.
In qualsiasi rivista, anche divulgativa, reperibile nelle edicole troviamo guide e corsi per l'apprendimento dell'uso dei computer quale strumento per svolgere attività di tipo quanto mai svariato (utilizzo di internet, applicazioni legate al mondo del multimedia e dei videogiochi, e così via).
Risulta d'altronde indispensabile al giorno d'oggi possedere competenze in ambito di Office Automation: ricerche importanti sottolineano da una parte i danni che derivano da un uso improprio delle tecnologie (il costo dell'ignoranza), dall'altra la necessità di sviluppare quanto più possibile le competenze a tutti i livelli , a partire da quello di base.
Importante in tal senso l'attività di **AICA**[9] volta a diffondere il programma di certificazione **ECDL**[10], riconosciuto a livello europeo e non solo, neutrale e indipendente dai fornitori di hardware e di software, e articolato a diversi livelli. Il programma della **Patente Europea del Computer** è sostenuto dalla Unione Europea, che l'ha inserito tra i progetti comunitari diretti a realizzare la Società dell'Informazione.

1.1.2.2. Utenti avanzati

È importante distinguere diversi livelli di utilizzo degli strumenti informatici, riconoscendo competenze di livello avanzato che si vanno diffondendo fra gli utenti dei computer. Un numero sempre maggiore di utenti utilizza il computer come strumento che esegue programmi avanzati, in diversi campi applicativi.
Parallelamente abbiamo le certificazioni di queste competenze di livello avanzato: ricordiamo il percorso **ECDL ADVANCED** che si riferisce ancora all'Office Automation, ma anche i percorsi *specialized*, ad esempio **CAD** (Computer Aided Design) e **GIS** (Geographic Information Systems) e Health (in campo paramedico).
Questi sono solo alcuni esempi, molti comunque i casi in cui competenze avanzate consentono di inserirsi in ambiti lavorativi specifici che possono rivelarsi molto promettenti ed interessanti (esempi grafica e multimedia, musica[11], formazione ed e-learning, e così via).

1.1.2.3. Professionisti

A livello più avanzato abbiamo i profili professionali **EUCIP** (European Certification of Informatics Professionals - Sistema Europeo delle Certificazioni per professionisti informatici)[12] riportati in tabella in ordine alfabetico.

---

[9] AICA: associazione culturale non profit che raccoglie i professionisti italiani dell'ICT
[10] Vedi nota 1 pagina
[11] Vedi http://www.mediamente.rai.it/mediamentetv/learning/corsi/0001c3_1.asp

Denominazione
Amministratore di sistemi informatici (IT Administrator)
Analista di Business
Analista di Sistemi Informativi
Analista Programmatore
Capo progetto di Sistemi Informativi
Consulente di Logistica e Automazione
Consulente di Soluzioni Aziendali
Consulente per la Sicurezza
Consulente vendita e applicazione Tecnologie Informatiche
Esperto di Applicazioni Web e Multimediali
Formatore IT
Progettista delle Telecomunicazioni
Progettista di Sistemi Informatici
Responsabile commerciale
Responsabile della Configurazione e del Centro Dati
Responsabile di Basi di Dati
Responsabile di Rete
Responsabile di Sistemi Informativi
Revisore di Sistemi Informativi
Sistemista multipiattaforma
Supervisore di un Centro di Assistenza
Tecnico di Collaudo e Integrazione di Sistemi

Modello EUCIP per la definizione delle competenze ICT

Ciascuno dei 22 profili EUCIP è definito da un insieme di attività proprie ed è caratterizzato da un proprio insieme di competenze (saper fare) definito nel corrispondente **syllabus**.
Il mercato ICT ha individuato profili professionali simili a quelli riportati, pur assegnando talvolta nomi diversi.

Come esempio riportiamo la descrizione del profilo dell'*Analista programmatore*[13]:

Un analista programmatore secondo lo standard EUCIP assume un ruolo tecnico di rilievo nella progettazione di sistemi informativi e deve essere molto efficace nella realizzazione e manutenzione di moduli software complessi, che tipicamente dovranno essere integrati in un più ampio sistema informativo. Sono possibili diverse specializzazioni, sia nel campo degli applicativi e dei servizi web, sia nel software a livello di sistema.
Questo profilo richiede un'esperienza lavorativa minima di 18 mesi in un ruolo professionale compatibile; se non possiede tale requisito, il candidato potrebbe essere certificato come Assistente Analista Programmatore.

Si riportano le competenze comportamentali fondamentali dell'*Analista programmatore*:

Il ruolo di analista programmatore richiede innanzitutto un atteggiamento mentale razionale in grado di pensare in modo concettuale e analitico, attenzione al dettaglio e un approccio fortemente orientato all'obiettivo, che porti al risultato attraverso soluzioni strutturate formulate in modo flessibile.
Un altro importante insieme di competenze è la capacità di comunicare e interagire in modo efficace (sia in forma orale che scritta) con colleghi e clienti: questo include una consapevolezza organizzativa e inter-funzionale generale, un buon approccio al lavoro di gruppo, efficienza nell'acquisizione delle informazioni, così come capacità di progettare, organizzare, prendere decisioni tecniche, fornire indicazioni e dare continuità.

---

[12] Il programma EUCIP, gestito in Italia da AICA, favorisce l'incontro tra domanda e offerta di lavoro qualificato nell'ambito dell'Information Technology fornendo uno schema di riferimento per le competenze professionali condiviso da imprese, enti formativi e Amministrazione pubblica e permettendo anche di certificare tali competenze in modo autorevole e indipendente dai fornitori. Per ulteriori informazioni http://www.eucip.it/

[13] http://www.eucip.it/standard-eucip/Il%20modello%20EUCIP.pdf

1.1.3. Sistemi informatici

1.1.3.1. Sistemi informatici

I sistemi informatici che vengono utilizzati nei vari contesti sono quanto mai diversificati: tutti hanno alcuni elementi comuni:
tecnologia elettronica
capacità di memorizzare ed elaborare dati in modo automatico, sulla base di opportuni programmi.
È consuetudine classificare i sistemi informatici in base alla tipologia come indicato nell'elenco a lato[14].

Tipologia
Mainframe
Minicomputer
Personal computer
Network computer

1.1.3.2. Personal Computer

Il personal computer ha segnato un momento fondamentale nella diffusione degli strumenti informatici, e quindi nel nostro modo di lavorare e di vivere: ricordiamo il primo numero del 1983 del settimanale "Time" che definisce il PC personaggio dell'anno[15].

1.1.3.3. Lan: Rete di PC

In genere si lavora con **reti di calcolatori** (insiemi di calcolatori connessi fra di loro)

Reti classificate per dimensione	
LAN	Local Area Network
WAN	Wide Area Network
Internet	a livello planetario

Da un punto di vista funzionale possiamo parlare di **architettura client/server**: ci si riferisce ad un sistema in cui abbiamo alcuni computer (server) che mettono a disposizione di altri computer (client) risorse o servizi. Esempio di risorsa condivisa: una stampante utilizzata da tutte le macchine di una rete locale come potrebbe essere il laboratorio di informatica di una scuola; esempio di servizi: la posta elettronica.

1.1.3.4. Internet

Internet (dal latino inter=fra e dall'inglese net=rete) è una rete di reti[16] con architettura client/server nata dal Ministero della Difesa degli Stati Uniti ai tempi della guerra fredda, apertasi agli usi civili in

[14] Si potrebbero inserire anche i palmari ed i cellulari, in generale ogni dispositivo con un processore, quinidi automobili, televisori, ecc.
[15] Vedi http://img.timeinc.net/time/images/covers/19830103_107.jpg
[16] E una rete WAN che utilizza la suite di protocolli TCP/IP e che si estende su tutto il nostro pianeta

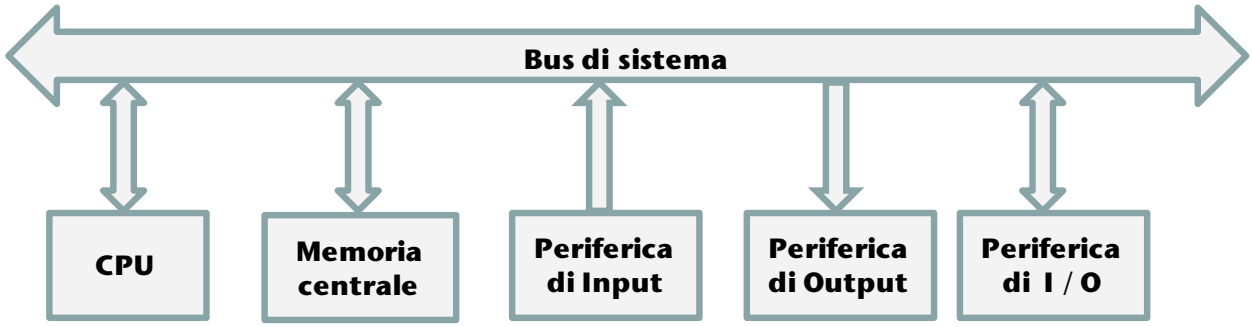
ambienti di ricerca, ed infine divenuta di uso comune grazie ad uno dei suoi servizi più apprezzati dagli utenti, il WWW (World Wide Web)[17].

1.1.4. Architettura dei sistemi di elaborazione

Principi essenziali per comprendere il funzionamento di un computer.

1.1.4.1. Architettura dei sistemi di elaborazione

Fin dagli anni 40 del secolo scorso Von Neumann progettò e descrisse l'architettura dei calcolatori in termini logico-funzionali.



Architettura di Von Neumann

Nel 1936 il matematico inglese Alan Turing (1912-1954), uno dei padri dell' informatica, progettò una macchina ideale che capace di risolvere problemi.

Un computer o è un dispositivo fisico costruito secondo i criteri della "Macchina di Turing"

Questo modello è ancora valido, anche se naturalmente con il tempo[18] c'è stata una evoluzione: gli attuali computer hanno una architettura che rappresenta un'estensione di quella di Von Neumann (ad esempio presenza di più processori, sia come processori dedicati ad esempio al calcolo numerico o alla grafica, sia come sistemi multiprocessor; gerarchia di memorie, ad esempio memoria cache).

L'espressione *architettura di un sistema di elaborazione* fa riferimento quindi all'insieme delle parti che lo costituiscono ed alle relative interconnessioni: possiamo distinguere due livelli:

Hardware: la parte fisica del sistema

Software: programmi eseguiti dal sistema

- Sistema operativo
- Software applicativo

[17] Nato nel 1991 al Cern di Ginevra http://public.web.cern.ch/public/Welcome.html come progetto che doveva sevire a permettere una comunicazione efficace fra gli scienziati ed i tecnici del Centro di ricerca; dal 1993 il WWW è stato aperto a tutti.

[18] Per una storia dello sviluppo dei computer vedi Mostra Museo dell' informatica http://www.dimi.uniud.it/cicloinf/mostra/index.html

1.1.4.2. Hardware

Da un punto di vista funzionale gli elementi della struttura di un elaboratore sono:

CPU (Central Processing Unit): esegue le istruzioni dei programmi (ad ogni istruzione corrispondono in effetti più operazioni elementari) e governa il funzionamento di tutte le unità

Memoria centrale: contiene i dati e le istruzioni del programma in esecuzione, è volatile

Periferiche di Input/Output: dispositivi per uno scambio di informazioni fra l'elaboratore e l'esterno (ad esempio tastiera, mouse, monitor, dispositivi di memoria di massa, modem, ecc).

Bus: canale per lo scambio di dati e segnali di controllo fra i diversi componenti

Per una introduzione vedi http://www.mediamente.rai.it/mediamentetv/learning/corsi/9911c1_1.asp, per riferimenti aggiornati gli aspetti tecnologici vedi ad esempio su http://www.hwupgrade.it o su altre riviste del settore.

1.1.4.3. Software

Proviamo a considerare, nella nostra esperienze di vita quotidiana, le situazioni in cui ci troviamo ad interagire con il software; segue una lista, non necessariamente completa né esaustiva, ma comunque esemplificativa:

- comunicazioni: uso del cellulare , accesso ad Internet (navigazione sul web, posta elettronica, social network, blog)
- svago : videogiochi, multimedia (fotografia, cinema, musica, ecc.)
- gestione attività domestiche: lavatrice, lavastoviglie, tutti *programmabili*!
- acquisti: carta di credito, bancomat, acquisti on-line
- servizi al cittadino: posta, banca

Di seguito si esamina specificatamente il software relativo ai sistemi di elaborazione.

1.1.4.4. Sistema operativo

Scopo del sistema operativo è quello di gestire le risorse hardware del computer e di consentire all'utente di accedere ad esse in modo trasparente (cioè non visibile all'utente: ad esempio scrivo un file su disco, senza conoscere i dettagli hardware e senza doverne quindi gestire il funzionamento) mediante una interfaccia semplice da utilizzare.

Architettura a livelli del software

Esso ha una struttura modulare, organizzata con una serie di livelli gerarchici: l'utente può accedere a tutte le funzionalità attraverso il livello più esterno, quello che costituisce l'interfaccia del sistema. In genere si utilizza una interfaccia grafica (GUI: Graphic User Interface) sia con Windows che con Linux, anche se in determinate situazioni i tecnici informatici preferiscono invece utilizzare l'interfaccia a riga di comando, scrivendo direttamente i comandi da eseguire.

1.1.4.5. Software applicativo

Ci si riferisce al software utilizzato per svolgere specifiche attività :

- livello di base: ad esempio blocco note, calcolatrice, paint, browser per navigare in internet
- livello avanzato: ad esempio per applicazioni di videoscrittura, CAD, applicazioni per ufficio, magazzino, ecc; videogiochi; multimedia; grafica, e così via

1.1.5. Sviluppo del software

Presentazione degli strumenti software utilizzati in questo testo per introdurre i principi della programmazione

1.1.5.1. Cosa significa sviluppare software ?

È chiaro che se è possibile utilizzare il computer come strumento *general purpouse* per attività di tipo diverso significa che il computer stesso esegue istruzioni specifiche che gli sono state fornite. Da chi? In che modo?

L'attività dei programmatori è quella di sviluppare software, cioè scrivere programmi che contengano le istruzioni per eseguire i diversi compiti: questo testo vuole aiutare ad acquisire le conoscenze e competenze necessarie.

Naturalmente servono anche gli strumenti. Se il programmatore deve scrivere le istruzioni da far eseguire al computer per svolgere un certo compito avrà bisogno di uno strumento di scrittura: a tale scopo in ambiente Windows si può utilizzare Blocco Note.

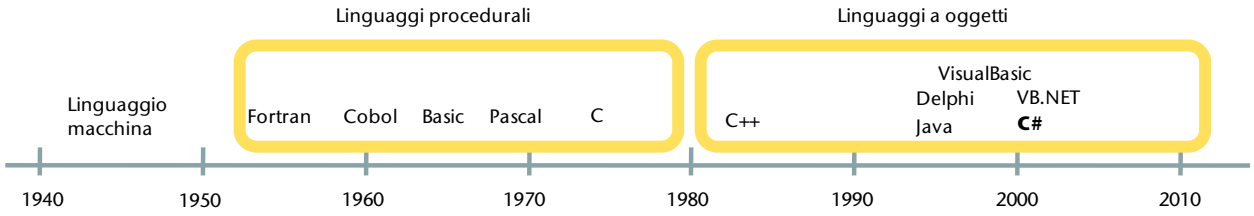
In quale linguaggio può il programmatore comunicare con il computer per fornire le istruzioni da eseguire? Semplice: in linguaggio macchina, cioè con un linguaggio che utilizza solo due simboli, lo zero e l'uno. Il computer non comprende altro, in quanto l'hardware è costituito di componenti elettronici attraverso i quali può solo passare la corrente (valore 1) o non passare (valore zero).

In realtà oggi lo sviluppatore (programmatore) di applicazioni utilizza linguaggi orientati non alla macchina, bensì al problema da risolvere.

Ci saranno altri programmi, scritti da altri programmatori, che si occupano di tradurre dal linguaggio utilizzato dal programmatore al linguaggio macchina (vedi Capitolo 3.1 Linguaggi di programmazione).

1.1.5.2. Storia dei linguaggi di programmazione

Si presenta una guida alla cronologia dei principali linguaggi di programmazione procedurali e ad oggetti.



Linguaggi di programmazione

linguaggi procedurali (imperativi): il programmatore impartisce all'elaboratore una serie di istruzioni da eseguire, specificando come si deve operare per risolvere il problema

- **Fortran** (FORmula TRANslation) → per applicazioni scientifiche – 1957 autore John Backus
- **Cobol** → (Common Business Oriented Language) per applicazioni gestionali - 1961
- **Basic** (Beginners All purpose Symbolic Instruction Code) → adatto a principianti, privilegia la facilità d'uso - 1964
- **Pascal** introduce la programmazione strutturata, utile per scopi didattici → creato da Niklaus Wirth - 1970
- **C** linguaggio efficiente utilizzato anche per la programmazione di sistemi operativi**,** →creato da Dennis Ritchie - 1972

Linguaggi ad oggetti

- **C++** introduce la OOP nell'ambiente dei programmatore in linguaggio C (programmazione orientata agli oggetti) creato da Bjarne Stroustrup - 1983
- **Java** rende migliore ed allo stesso tempo più semplice la programmazione ad oggetti, linguaggio multipiattaforma, Sun Microsystems, 1996
- **Delphi**
- **Visual Basic**
- **C#**
- **VB.NET**

Vi sono altri tipi di linguaggi, ricordiamo in particolare:

linguaggi logici e funzionali → il programma valuta una serie di funzioni matematiche (fra i linguaggi attuali ad esempio Linq)

linguaggi dichiarativi → vengono formulate delle regole che descrivono il risultato che si vuole ottenere, non il procedimento da seguire (ad esempio SQL)

In questo testo utilizzeremo il linguaggio C#. Tale scelta dovuta a due motivazioni principali:

- importanza del C# nel panorama dei linguaggi di programmazione e suo ruolo nell'ambito della piattaforma .NET
- validità didattica in quanto garantisce una buona *pulizia concettuale* e consente sia di mostrare le basi della programmazione procedurale, sia di introdurre con semplicità all'uso degli oggetti ed alla realizzazione di applicazioni in ambiente Windows e di applicazioni WEB.

1.1.5.3. .NET Framework 3.5

Il programmatore C# dispone di strumenti che migliorano il suo lavoro: di seguito ne indichiamo alcuni, con lo scopo sia di presentare l'ambiente di lavoro con cui andremo ad operare, sia di contestualizzarlo nel panorama attuale.

Esistono diverse piattaforme: in questo testo si fa riferimento a Microsoft, scelto come punto di riferimento per l'ampia diffusione attuale.

Il **Microsoft® .NET® Framework** è un componente di Windows che fornisce al programmatore un ambiente di programmazione orientato agli oggetti ed è a sua volta costituito da due componenti principali:

Common Language Runtime (CLR): ambiente per l'esecuzione dei programmi[19]

libreria di classi .NET Framework: è un insieme di strumenti di cui il programmatore può disporre liberamente e che può quindi utilizzare nei propri programmi.

Tale piattaforma supporta un elevato numero di linguaggi di programmazione ed è predisposta verso l'utilizzo di formati aperti per lo scambio dei dati (XML).

La versione 3.5 contiene tutte quelle precedenti, è liberamente scaricabile[20] e si può installare su Windows Server 2003, Windows Server 2008, Windows Vista e Windows XP.

1.1.5.4. Microsoft® Visual Studio® 2008 Express Edition

Il programmatore utilizza abitualmente **ambienti di sviluppo** che consentono non solo di scrivere i programmi, ma anche di gestire tutte le fasi di sviluppo del software (ad esempio il debug, oppure lo sviluppo rapido di applicazioni RAD).

Visual Studio 2008 è una suite di strumenti di sviluppo che Microsoft propone per gli sviluppatori di qualsiasi livello, siano essi alle prime esperienze di programmazione, o siano al contrario professionisti: questo è uno dei motivi per cui si ritiene vantaggioso utilizzare tale ambiente.

Molto importante è il fatto che sia possibile sviluppare sia applicazioni desktop (che dovranno quindi essere eseguite su PC), sia applicazioni web o su dispositivi portatili: questo consente di far tesoro delle conoscenze e competenze acquisite e di riutilizzarle in nuovi contesti applicativi durante tutto il percorso di studi e di lavoro.

Sono disponibili versioni a pagamento (ad esempio Professional Edition) ma anche versioni Express totalmente gratuite, per diversi linguaggi di programmazione, che possono essere liberamente scaricate ed utilizzate.

1.1.5.5. Linguaggio di programmazione C#

Il linguaggio di programmazione C# (si legge C sharp) è stato sviluppato da Microsoft come linguaggio per il Framework .NET e poi approvato come standard ECMA[21].
C#, progettato da Anders Hejlsberg [22], è un linguaggio moderno, orientato agli oggetti , che da un lato eredita i punti di forza di altri linguaggi importanti quali C++ e Java, dall'altro si avvale di tutta la potenza della architettura .NET.

---

[19] In termini più precisi parliamo di applicazioni managed , cioè gestite appunto dal CLR

[20] Al momento in cui si scrive questo testo il link preciso è http://www.microsoft.com/downloads/details.aspx?displaylang=it&FamilyID=333325fd-ae52-4e35-b531-508d977d32a6#QuickInfoContainer, altrimenti si faccia riferimento del sito Microsoft Italia riportato nella sitografia

[21] European Computer Manufacturers Association, associazione che si occupa di definire in ambito ICT standard dei sistemi di informazione. Per maggiori informazioni http://www.ecma-international.org/

[22] Precedentemente alla Borland capo architetto del progetto Delphi, e prima ancora del Turbo Pascal

Come ambiente per lo sviluppo in linguaggio C# in questo testo si fa riferimento a **Visual C# 2008 Express Edition** che ci si può liberamente procurare effettuando un download da http://www.microsoft.com/express/download/ . Naturalmente è necessario anche il .NET Framework, anch'esso libero (vedi paragrafo 1.1.5.3 .NET Framework 3.5).

1.1.5.6. Soluzioni alternative a Microsoft per lo sviluppo con linguaggio C#

È possibile programmare in linguaggio C# utilizzando un ambiente integrato di sviluppo gratuito e open source con licenza GNU Lesser GPL, alternativo a Visual Studio . Ci si riferisce a **SharpDevelop 3.0**, scritto in C#, che viene eseguito in ambiente Windows con .NET Framework: è vero che la maggior parte dei programmatori C# utilizza Visual Studio, ma può essere interessante provare questo ambiente, anche per un confronto.

Per ulteriori informazioni vedi http://www.sharpdevelop.com/OpenSource/SD/ .

Importante osservare che, oltre alla versione .NET di Microsoft, esiste un **progetto open source** che ha rilasciato **Mono 2.0,** implementazione libera della piattaforma .NET disponibile per i sistemi operativi **Linux** e **Mac OS** (pressoché compatibile con .NET 3.0 e con il supporto completo di C# 3.0): ciò significa che sono compatibili, senza modifiche, le applicazioni scritte per essere eseguite sui diversi sistemi operativi. Per ulteriori informazioni http://www.mono-project.com/Main_Page .

1.1.5.7. Licenze software

Esistono diverse filosofie di produzione e utilizzo del software che vengono di seguito brevemente richiamate.

commerciale	il software viene acquistato con licenza d'uso: l'acquirente non è proprietario del codice sorgente (cioè del codice che contiene le istruzioni scritte dal programmatore) ma può solo limitarsi ad utilizzare il software che viene resi disponibile in versione eseguibile (cioè direttamente con le istruzioni in linguaggio macchina); non è permesso duplicare, modificare, ridistribuire il software
shareware	è disponibile l'eseguibile, ma non il sorgente; la licenza d'uso prevede un piccolo importo, una scadenza temporale o alcune limitazioni d'uso; è permesso distribuire il programma (chi lo riceve è soggetto agli stessi obblighi)
freeware	l'eseguibile è gratuito ma non viene dato il codice; nessuna licenza d'uso, nessuna limitazione alla distribuzione (ciò può favorire strategie di marketing e/o tentativi di tentativi di imposizione di standards)
Software libero[23] GPL	viene reso disponibile l'eseguibile e anche il codice sorgente. è consentita la possibilità di modifica dei sorgenti; è possibile duplicare il software; chiunque utilizzi questo software è vincolato alle stesse regole. Questo deriva dall'idea di Richard Stallman che nel 1985 fonda la FSF (Free Software Foundation) che definisce la GPL (General Public Licence) vedi http://www.fsf.org/ , http://www.gnu.org/home.it.html
open source	Il termine Open Source del 1998 introduce varianti per cui ogni software che risponde alla GPL della FSF è Open Source, ma non necessariamente il contrario

[23] Si parla anche, più in generale, di contenuti liberi, riferendosi non solo al software, ma anche a testi, brani musicali , ecc. Per maggiori informazioni vedi http://www.copyleft-italia.it/ .

1.1.5.8. Quali saranno gli ambienti ed i linguaggi di domani?

In questo testo si farà riferimento a specifici strumenti software, in quanto è utile non limitarsi ad introdurre i principi della programmazione in modo teorico, ma consentire anche di acquisire abilità pratiche e sviluppare esperienze personali concrete indispensabili per un vero apprendimento.
La scelta che si è effettuata circa gli strumenti software di riferimento non è comunque limitativa, in quanto tutte le conoscenze e competenze sviluppate possono essere agevolmente trasferite ed utilizzate anche su altre piattaforme.

È importante acquisire principi di base: gli strumenti con i quali ci troveremo a lavorare i prossimi anni saranno dettati sia dallo sviluppo tecnologico, sia da aspetti economici e strategie commerciali. Chi si occupa di informatica deve mostrare interesse e propensione all'aggiornamento continuo, ma non bisogna soccombere di fronte ad una realtà che cambia con velocità davvero impressionante. Si tratta non tanto di correre, e tanto meno di inseguire quanto di nuovo si affaccia man mano nel mondo dell'IT, quanto soprattutto di riuscire ad avvalersi di solide conoscenze e competenze di base. In questo libro .NET Framework 3.5 e Visual Studio 2008 sono *strumenti* che ci consentono di entrare nel modo della programmazione.

HAI IMPARATO CHE...

1. Lavorare nel campo dell' informatica è interessante
2. Stai per imparare a programmare il computer

2. Algoritmi + Dati

Thomas Kuhn afferma "...ad esempio, lo studente di dinamica newtoniana scopre il significato di termini come 'forza', 'massa', 'spazio' e 'tempo' non tanto sulla base di definizioni incomplete, sebbene talvolta utili, contenute nel suo manuale, ma osservando e partecipando alla applicazione di questi concetti nella soluzione dei problemi"[24]

Nel nostro caso lo studente deve sviluppare la capacità di affrontare problemi, per ora semplici, ed individuarne la soluzione; di seguito vedremo come utilizzare il computer per eseguire le operazioni che noi abbiamo scoperto essere utili per risolvere il nostro problema.

Vengono qui indicati alcuni percorsi, fondamentale è un atteggiamento attivo dello studente per acquisire i concetti e sviluppare le competenze che costituiscono il bagaglio di chi lavora nel campo dell' informatica e della programmazione.

Metodologia di soluzione dei problemi

Algoritmi

Diagrammi a blocchi strutturati

Soluzione di problemi semplici

[24] T.S.**Kuhn**: La struttura delle rivoluzioni scientifiche, Einaudi, 1969

2.1. METODOLOGIA DI SOLUZIONE DEI PROBLEMI

PREREQUISITI
Saper Leggere
OBIETTIVI
Acquisire una metodologia di soluzione dei problemi

2.1.1. Un problema da risolvere

Per sviluppare abilità nella soluzione dei problemi è necessario acquisire un buon metodo di lavoro, che si basa su alcuni principi: lettura puntuale del testo, con particolare attenzione a identificare i dati che sono a disposizione e i risultati richiesti al fine di progettare una soluzione corretta ed efficiente; infine verifica della bontà della soluzione individuata: cominciamo con un esempio.

Oggi l'insegnante di informatica procede alle prime verifiche e, per ciascun alunno interrogato, assegna un voto. Si è interessati a conoscere il voto migliore ottenuto dagli alunni.

Il problema da risolvere è abbastanza semplice: proviamo a riflettere. Non è stato specificato il numero di alunni che vengono interrogati, d'altra parte può essere che non si conosca a priori questo dato (dipende probabilmente dal tempo necessario per svolgere le diverse interrogazioni).

Quando viene assegnato il primo voto, possiamo senz'altro affermare che, al momento, è il voto migliore; è poi da vedere in seguito se gli alunni interrogati riescono a superare la prestazione del loro primo compagno.
In effetti, man mano che si conclude un'altra interrogazione è sufficiente confrontare il voto ottenuto con quello che rappresentava il massimo ottenuto dai compagni precedenti, per vedere se si ottiene un nuovo massimo (il voto del vincitore di questa particolare competizione).

Il problema sembra risolto, basta fare attenzione a quando l'insegnante comunica il nuovo voto assegnato, e ripetere sempre lo stesso confronto. Ma: come facciamo sapere quando sono terminate le interrogazioni? In questo caso non dobbiamo continuare a rimanere in attesa di conoscere un nuovo voti! Stabiliamo questo accordo: l' insegnante in questo caso ci comunica il valore zero. In effetti i voti hanno valori compresi fra uno e dieci, quindi lo zero non è sicuramente un voto, rappresenta per noi il segnale di fine.

Proviamo a schematizzare la situazione, per vedere se la soluzione che abbiamo individuato possa essere corretta.

A tal proposito si predispone una tabella[25] in cui registrare i vari voti, al fine di tenere traccia precisa dell'andamento delle interrogazioni (ogni riga della tabella rappresenta una particolare azione).

Descrizione Azioni svolte	Voto assegnato	Voto massimo	Risultato finale
Viene effettuata la prima interrogazione ed assegnato il voto	6		
Il voto della prima interrogazione è per ora il massimo fra i voti assegnati (lo riporto nella colonna corrispondente)		6	
Viene effettuata un'altra interrogazione con relativo voto	5		
Confronto il voto dell'ultima interrogazione con il voto massimo fin qui ottenuto: il nuovo risultato è peggiore!			
Viene effettuata un'altra interrogazione con relativo voto	7		
Confronto il voto dell'ultima interrogazione con il voto massimo fin qui ottenuto: il nuovo risultato è migliore quindi devo annotarlo nella colonna che riporta il voto massimo!		7	
Viene effettuata un'altra interrogazione con relativo voto	6		
Confronto il voto dell'ultima interrogazione con il voto massimo fin qui ottenuto: il nuovo risultato è peggiore!			
Viene effettuata un'altra interrogazione con relativo voto	8		
Confronto il voto dell'ultima interrogazione con il voto massimo fin qui ottenuto: il nuovo risultato è migliore quindi devo annotarlo nella colonna che riporta il voto massimo!		8	
Viene effettuata un'altra interrogazione con relativo voto	6		
Confronto il voto dell'ultima interrogazione con il voto massimo fin qui ottenuto: il nuovo risultato è peggiore!			
L'insegnante comunica il valore zero: ciò significa che le interrogazioni sono concluse	0		
Possiamo dichiarare qual è, per oggi il voto migliore: lo troviamo nella colonna apposita e, per maggiore evidenza, lo trascriviamo nella colonna dei risultati.			8

[25]Nella colonna "Descrizione azioni svolte" i diversi colori mettono in evidenza le operazioni che si ripetono (ad ogni specifica operazione corrisponde sempre lo stesso colore di sfondo del testo)

Osserviamo il risultato ottenuto quale valutazione massima conseguita dagli studenti, nel nostro caso il valore 8 come si ricava dall'ultima riga della tabella, in corrispondenza della colonna del risultato finale. Dobbiamo sempre chiederci: il risultato ottenuto corrisponde alle aspettative? Il valore è corretto? In questo caso la risposta è positiva.

Consideriamo un problema simile.

L'insegnante di inglese procede alle prime verifiche interrogando 6 alunni, a ciascuno dei quali assegna un voto. Si è interessati a conoscere il voto migliore ottenuto dagli alunni.

Sono evidenti le analogie con il caso precedente, ma osserviamo una differenza importante: conosciamo il numero esatto di interrogazioni, quindi possiamo modificare la tabella introducendo un indicatore con valori progressivi a identificare quale alunno viene interrogato (evitando quindi di chiudere l'elenco dei voti con lo zero, come abbiamo fatto prima).

	N° progr. interrogazione	Voto assegnato	Voto massimo	Risultato finale
Viene assegnato un voto	1	4		
Essendo la prima interrogazione, è per ora il massimo dei voti ottenuti dagli alunni			4	
Viene assegnato un voto	2	6		
Aggiorno il valore del voto massimo ottenuto			6	
Viene assegnato un voto	3	5		
Il voto massimo rimane invariato				
Viene assegnato un voto	4	6		
Il voto massimo rimane invariato				
Viene assegnato un voto	5	7		
Aggiorno il valore del voto massimo ottenuto			7	
Viene assegnato un voto	6	6		
Il voto massimo rimane invariato				
Terminate le 6 interrogazioni possiamo dichiarare qual è, per oggi il voto migliore: lo troviamo nella colonna apposita e, per maggiore evidenza, lo trascriviamo nella colonna dei risultati.				7

Gli esempi mostrati indicano come per risolvere un problema sia necessario scoprire una strategia risolutiva che, partendo dai dati che possediamo inizialmente, ci consenta, mediante operazioni opportune, di raggiungere il risultato che ci siamo posti.

In ogni caso è necessario ricorrere alla fantasia di chi affronta il problema, non esistono soluzioni a priori. È comunque vero che non sempre è necessario scoprire tutto da soli, anzi è fondamentale appoggiarsi all'esperienza di chi ha già affrontato problemi simili per avvalerci non tanto del risultato, spesso neanche della specifica strategia risolutiva, ma invece sempre del metodo di lavoro utilizzato per scoprire la soluzione[26].

Un informatico deve cercare sempre di trovare una strategia di soluzione originale ai problemi che si trova ad affrontare, non può pensare di limitarsi a seguire meccanicamente soluzioni già note, perché studiate in modo diligente o magari copiate da qualche sito internet: tale compito è demandato al computer.

Il lavoro di chi si occupa di informatica non è banale, ricordiamoci sempre che la conoscenza dei metodi risolutivi di alcuni problemi classici ci fornisce comunque il bagaglio culturale ed il metodo per scoprire la soluzione dei problemi nuovi che ci troviamo ad affrontare.

PAUL K. FEYERABEND
Le regole metodologiche devono essere adattate alle circostanze e reinventate sempre di nuovo. Ciò aumenta la libertà, la dignità e la speranza di successo.

2.1.2. Processo di sviluppo del software

Il corso di Informatica ha come fine principale quello di mettere il Perito in Informatica in grado di affrontare (dall'analisi fino alla documentazione) la soluzione di un problema, posto dalla richiesta di un ipotetico committente, scegliendo le metodologie e gli strumenti software: risulta quindi fondamentale conoscere un metodo di lavoro adeguato.

2.1.2.1. Fasi dello sviluppo del software

Chi lavora nel campo dell'informatica e ricerca la *soluzione di un problema mediante l'uso di un calcolatore* non deve solo occuparsi della attività specifica svolta al computer e basata sullo sviluppo del codice, ma è importante che segua complessivamente un buon metodo di lavoro[27] articolato in fasi differenti:

---

[26] Ricordiamo il testo di Donald Knuth, **The Art of Computer Programming**, considerato un riferimento importante nella comunità degli informatici. Rifletti sul titolo!

[27] Metodo è un insieme di indicazioni che riguardano il modo migliore di svolgere un' attività.
Socrate nei "dialoghi platonici" risulta già cosapevole della relazione tra la validità di una conoscenza e il modo in cui essa viene ottenuta e pone l' attenzione su due tipi di prescrizioni: 1) evitare l' errore, 2)provare ad arrivare alla conoscenza. Un atteggiamento analogo appartiene a chi cerca di utilizzare un buon metodo di lavoro per risolvere un problema.

Fase di lavoro	Descrizione attività	Documenti prodotti	verifica
analisi del problema cosa ?	individuare cosa fare per per risolvere il problema[28]	Relazione in linguaggio naturale (italiano)	Approvazione del committente
Algoritmo come?	dettagliare il procedimento risolutivo con operazioni semplici e non ambigue[29]	Tabella descrizione variabili + Diagramma a blocchi	Tabella di traccia
codifica del software, testing	Programmazione, collaudo e validazione	progetto software manuale di documentazione	
messa in esercizio	distribuzione, manutenzione e revisione, aggiornamento	versioni di aggiornamento	

INGEGNERIA DEL SOFTWARE
Disciplina che studia diverse metodologie per lo sviluppo del sofware, proponendo anche approcci diversi rispetto alla sequenza di esecuzione delle varie fasi di sviluppo qui presentata.
L' obiettivo è produrre sistemi software nei tempi e costi preventivati, assicurando naturalmente una qualità accettabile.

In ogni fase di lavoro si deve provvedere a verificare la correttezza del proprio operato, considerando sempre che ogni fase eredita tutti gli errori delle fase precedenti, e ne può aggiungere dei propri. Eventuali errori non corretti nelle prime fasi e che vengono messi in evidenza solo inseguito obbligano ad annullare tutto il lavoro fatto!

2.1.3. Metodologia di soluzione dei problemi

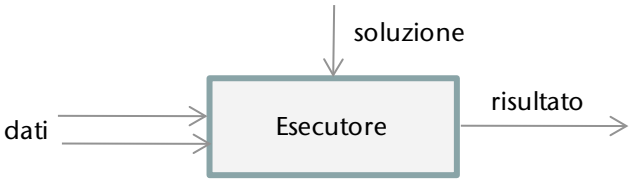
Per risolvere un problema con l'uso del computer è necessario utilizzare un buon metodo. Al fine di evitare ambiguità e quindi incomprensioni una breve introduzione per definire il significato di alcuni termini importanti, quindi passiamo ad esaminare le attività relative alle diverse fasi di lavoro individuate nel capitolo precedente.

2.1.3.1. Definizioni

Problema: situazione che presenta una difficoltà da superare
Soluzione: procedimento che permette di risolvere un problema, o meglio una *classe di problemi*, cioè un insieme di problemi che si differenziano solo per i valori dei dati (e quindi dei risultati)
Esecutore: chi esegue le operazioni per risolvere il problema; può essere un uomo o una macchina (ad esempio il computer)

[28] Cioè per passare dai dati iniziali ai risultati finali che vengono richiesti dal committente
[29] Questo passaggio è necessario per poter poi utilizzare il calcolatore come esecutore delle istruzioni stesse e quindi come strumento che ci aiuta nella soluzione del problema

Dati: valori che devono essere forniti in modo che operando su di essi si possa risolvere uno specifico problema (valori di input); in generale risorse che ci vengono date e che quindi, essendo a nostra disposizione, che si possono utilizzare per risolvere il problema;
Risultato: valore che si ottiene partendo dai dati di ingresso e seguendo il procedimento per risolvere un determinato problema (valore di output)



Esempio:
Problema: calcolare la somma di due valori utilizzando una calcolatrice .
Dati: i due valori da sommare, ad esempio 2 e 3
Soluzione: scrivere il primo valore, premere il pulsante + , scrivere il secondo valore, premere il pulsante =, leggere il dato visualizzato
Risultato: valore 5
Esecutore: chi esegue le operazioni per risolvere il problema; può essere un uomo o una macchina (ad esempio il computer)

2.1.3.2. Analisi

Il lavoro di analisi consiste nello specificare con precisione cosa si deve fare per risolvere il problema, cioè quali sono le richieste, cosa deve essere realizzato, cosa si aspetta il committente; si produce un documento in cui in linguaggio naturale si specifica **cosa** intendiamo fare per ottenere il risultato richiesto.

Per effettuare una buona analisi bisogna *comprendere il problema* e quindi innanzi tutto leggere con attenzione il testo del problema, oppure discutere in modo puntuale con chi ci sottopone il problema da risolvere (committente), e verificare di aver ben compreso la realtà specifica a cui ci si riferisce e ciò che si vuole ottenere.

Si tratta di riconoscere quali sono gli elementi eventualmente superflui che ci vengono forniti e trascurarli , focalizzando la nostra attenzione sui soli elementi di interesse; d'altra parte verificare di possedere tutti gli elementi necessari ad elaborare una strategia risolutiva del problema proposto.
In caso contrario si tratta di formulare nostre ipotesi integrative, da sottoporre a verifica con il committente; talvolta può essere opportuno riformulare il problema che ci viene posto.
Attenzione: non si deve trascurare né contraddire nulla di ciò che ci viene specificato dal committente, eventualmente sarà anzi necessario studiare la realtà cui ci si riferisce per avere conoscenze adeguate (non posso risolvere un problema di geometria se non conosco le formule). In questo caso devo innanzi tutto studiare geometria; nello stesso modo non posso gestire un software per un ufficio se non conosco bene il lavoro che in quell'ufficio si svolge, e così via.

Nella analisi del problema è fondamentale individuare i dati[30] che ci vengono forniti, perché anche da essi dipende la strategia risolutiva che dovremo elaborare.

[30] Attenzione, ci si riferisce qui al tipo di dati, cioè alla categoria, non agli specifici valori: ad esempio 3 e 65 sono valori di tipo intero, "pippo" non è un valore di tipo intero!

Problema→ Cercare un numero di telefono

Il testo è troppo generico, prima di risolvere questo problema dobbiamo capire il contesto. Dove posso reperire questo dato? Ho a disposizione una agendina? Se sì. è organizzata secondo un certo ordine? Quale? Oppure il dato deve essere reperito da un elenco su carta? O ancora : da Internet?
È chiaro che a seconda di cosa ci viene fornito il procedimento per risolvere il problema sarà ben differente!

Problema→ Calcolare il perimetro e l' area di un rettangolo

È chiaro quale è il risultato da ottenere, ma nulla ci viene detto sui dati. In questo caso possiamo provare a formulare delle ipotesi, ragionevolmente possiamo chiedere che ci vengano forniti i valori della misura della base e dell'altezza

Problema→ Calcolare la misura dell' area di un dato cerchio .

Anche in questo caso è chiaro quale è il risultato da ottenere, ma ancora nulla ci viene detto sui dati.
Qui possiamo formulare diverse ipotesi e sarebbe meglio poterci rivolgere al committente per discuterle.
In particolare, ci potrebbe venire fornita la misura del raggio, ma non è l'unica opzione possibile. Ci potrebbe anche essere fornito il diametro, oppure la posizione del centro e di un punto sulla circonferenza, o infine la posizione di tre punti sulla circonferenza. Anche in questo caso a seconda dei dati che utilizzo è differente la soluzione, cioè il procedimento da seguire per ottenere il risultato (misura dell' area) .

2.1.3.3. Tabella delle variabili

Si è visto che nella strategia per risolvere un problema i dati giocano un ruolo molto importante: per generalizzare il problema (e risolvere quindi tutta una classe di problemi) è necessario riferirsi non a specifici valori, ma in generale a **variabili** che rappresentano i dati, in modo analogo a quanto si fa in matematica.

Se ci riferiamo ad esempio al problema del calcolo del perimetro ed area del rettangolo esaminato nel paragrafo precedente le variabili possono essere quelle indicate nella tabella qui di seguito:

Tabella di descrizione delle variabili

Nome	Descrizione	Tipo di dato	Input/Output
L1	Primo lato del rettangolo	numero intero	Input
L2	Secondo lato del rettangolo	numero intero	Input
P	Perimetro	numero intero	Output
A	Area	numero intero	Output

Il **tipo di dato** è associato ai valori ed alle operazioni che intendiamo eseguire su di essi: in questo caso si ipotizza quindi che i valori siano tutti di tipo intero (non sarà valido ad esempio il valore 3,2).

Se invece ci riferiamo al problema del calcolo dell' area del cerchio conoscendo le coordinate del centro e di un punto sulla circonferenza si avrà la situazione descritta nella tabella seguente. Da

osservare che la variabile raggio, utile per il calcolo dell' area, non rappresenta né un dato in input, né un risultato: si indica quindi come variabile di lavoro, in quanto ci serve appunto per il nostro lavoro.

Tabella di descrizione delle variabili

Nome	Descrizione	Tipo di dato	Input/Output/Lavoro
XC	Ascissa del centro	Numero reale	Input
YC	Ordinata del centro	Numero reale	Input
XP	Ascissa punto sulla circonferenza	Numero reale	Input
YP	Ordinata punto sulla circonferenza	Numero reale	Input
R	Raggio	Numero reale	Lavoro
A	Area	Numero reale	Output

2.1.3.4. Algoritmo

Si da qui ora una presentazione informale ed intuitiva[31] : algoritmo è la definizione dettagliata e precisa delle operazioni elementari da compiere per risolvere un problema.

Mentre nell'analisi di un problema cerchiamo di specificare cosa si può fare per risolverlo, in fase di **algoritmo** è necessario far capire **come** bisogna fare per mettere in pratica le idee che abbiamo avuto, dettagliando le operazioni ed evitando ogni ambiguità.

Esempi di problemi per i quali si possono fornire gli algoritmi sono:

- sostituire le cartucce della stampante
- calcolare l'area di un rettangolo conoscendo le misure dei lati
- calcolare il Massimo Comun Divisore fra due numeri interi

Non si riesce invece a fornire algoritmi per risolvere problemi tipo i seguenti:

- Dove trascorrere le vacanze questa estate?
- Cucinare la zuppa di pesce

In questi casi si possono dare suggerimenti, ma non è possibile dare un elenco preciso di istruzioni.
Nel primo caso si potrebbe aiutare un persona invitandola a riflettere quali sono le proprie aspettative per le vacanze, se pensa ad un viaggio lontano, oppure se preferisce attività sportive, oppure dedicarsi al riposo, e così via. Ciascuno di noi potrebbe esprimere opinioni differenti!
Nel secondo caso il suggerimento potrebbe essere quello di procurarsi del pesce fresco, ma cosa significa esattamente ciò? Ciascuno di noi potrebbe avere una idea differente sulla opportunità o meno di acquistare *quel* particolare pesce.

Non tutti i problemi si possono risolvere con l'uso di un calcolatore! Esso è un semplice esecutore, quindi ci può essere utile solo per risolvere quei problemi per i quali *noi* siamo in grado di individuare un procedimento risolutivo che si possa eseguire in modo automatico. Si parla in questo caso di **problemi calcolabili**.

[31] Per una definizione formale vedi unità successiva

2.1.3.5. Tabella di traccia

Una volta individuato un metodo per risolvere un dato problema dobbiamo sempre verificarne la correttezza. A tal proposito possiamo chiedere ad un nostro amico di provare a seguire tutte le operazioni da noi individuate e verificare se il risultato ottenuto è quello atteso.
Le istruzioni devono essere formulate in modo non ambiguo, così che il nostro amico le possa eseguire senza alcun nostro intervento: deve cioè lavorare in modo meccanico, come se fosse un computer (se non abbiamo un amico che si presti a questo compito possiamo, o meglio dobbiamo, eseguire noi stessi la verifica).

Problema→ Verificare la correttezza del seguente algoritmo (derivato dal metodo di Euclide, matematico greco vissuto fra 11 300 ed il 400 a.c.) per il calcolo del massimo comun divisore fra due numeri rappresentati dalle variabili M ed N.

1	se M>0 vai alla riga 2, altrimenti vai alla riga 5
2	se N>M allora scambia M con N
3	Sottrai N da M e assegna ad M il valore della differenza
4	vai alla riga 1
5	N è il MCD cercato

Algoritmo per il calcolo del Massimo Comun Divisore

Per sapere se le istruzioni che ci vengono date sono corrette non ci resta che provare: scegliamo due numeri qualsiasi, ad esempio M=20 ed N=8 come valori di INPUT, sapendo che in questo caso dovremmo ottenere il valore 4 come risultato.

Utilizziamo una **tabella** per tenere **traccia** del nostro lavoro, eseguendo passo per passo le operazioni indicate dall'algoritmo proposto:

passo di esecuzione	riga algoritmo	**Descrizione Azioni svolte**	**M**	**N**	**MCD**
		indico nelle colonne a fianco i valori di INPUT	20	8	
1	1	Verifico che M>0 (in quanto M ha il valore iniziale 20) quindi vado alla riga 2			6
2	2	verifico che non è necessario scambiare N con M quindi passo alla riga successiva			
3	3	calcolo M-N, cioè in questo caso 20-8 ed assegno il valore ottenuto 12 alla variabile M, riportandolo nella colonna qui a fianco quindi passo alla riga successiva	12		
4	4	ritorno alla riga 1			
5	1	Verifico che ancora M>0 (in quanto M ha ora il valore 12) quindi vado alla riga 2			

passo di esecuzione	riga algoritmo	**Descrizione Azioni svolte**	**M**	**N**	**MCD**
6	2	verifico che non è necessario scambiare N con M quindi passo alla riga successiva			
7		calcolo M-N, cioè in questo caso 12-8 ed assegno il valore ottenuto 4 alla variabile M,quindi passo alla riga successiva	4		
8	4	ritorno alla riga 1			
9	1	Verifico che ancora M>0 (in quanto M ha ora il valore 12) quindi vado alla riga 2			
10	2	Verifico che N>M, quindi li scambio e passo alla riga successiva	8	4	
11	3	calcolo M-N, cioè in questo caso 8-4 ed assegno il valore ottenuto 4 alla variabile M,quindi passo alla riga successiva	4		
12	4	ritorno alla riga 1			
13		Verifico che ancora M>0 (in quanto M ha ora il valore 4) quindi vado alla riga 2			
14	2	Verifico che ora M=N, quindi non sono da scambiare e passo alla riga successiva			
15	3	calcolo M-N, cioè in questo caso 4-4 ed assegno il valore ottenuto 0 alla variabile M,quindi passo alla riga successiva	0		
16	4	ritorno alla riga 1			
17	1	ora m=0, quindi vado alla riga 5			
18	5	riporto il valore di N nella colonna del Massimo Comun divisione			4

Osservo che il risultato ottenuto è proprio quello atteso, ciò mi fa pensare che l'algoritmo fornito possa essere corretto.

Naturalmente posso affermare ciò se riesco ad ottenere il risultato corretto per qualsiasi coppia di dati di input in quanto l'algoritmo deve risolvere non il solo problema del calcolo dl MDC fra 20 e 8, ma una **classe di problemi**, cioè tutti quei problemi che hanno lo stesso procedimento risolutivo e che si differenziano solo per i valori dei dati di INPUT. Ma allora come sapere se l'algoritmo è davvero corretto? Non si possono certo fare le prove per infinite coppie di valori!

Si tratta di eseguire delle verifiche significative su casi di esempio, come abbiamo fatto prima, cercando di scoprire eventuali casi particolari: per esempio cosa succede se uno dei due valori è negativo?

2.1.3.6. Codifica , collaudo e validazione

Se abbiamo un problema per il quale riusciamo a descrivere la soluzione in termini di operazioni dettagliate e non ambigue possiamo pensare di far eseguire queste stesse operazioni ad un computer, pur di riuscire a comunicarle con un linguaggio adeguato.

L'espressione **codifica del software** indica la fase di lavoro in cui viene scritto il programma che dovrà essere eseguito dal computer: si utilizza un linguaggio di programmazione ad alto livello, che dovrà poi essere tradotto in linguaggio macchina per poter essere eseguito.

Di seguito si provvede ad effettuare sul software prodotto dei test con dati di prova, a correggere eventuali errori, sino ad arrivare al collaudo ed alla validazione.

2.1.3.7. Distribuzione e manutenzione del software

In questa fase si ha la messa in esercizio, cioè la distribuzione del software che sarà disponibile per l'utente finale. Il programmatore può ancora intervenire con eventuali revisioni al fine di appontare migliorie con l'eventuale rilascio di versioni di aggiornamento.

2.1.3.8. Documentazione

Questo paragrafo è stato inserito alla fine per sottolineare il fatto che talvolta, sbagliando, si lascia proprio al termine del lavoro la stesura della documentazione; in effetti si tratta di una attività che va svolta durante tutte le fasi di lavoro: ogni fase ha dei prodotti specifici relativi al lavoro che si è svolto in essa e che devono essere corredati dalla relativa documentazione.
Ciò è particolarmente importante in quanto la documentazione prodotta da una fase di lavoro costituisce l'input per la fase di lavoro successiva: si pensi ad esempio al passaggio dall'analisi del problema alla formulazione dell'algoritmo, ed infine alla codifica.

Si distingue fra documentazione interna (tecnica, rivolta alla comunicazione fra coloro che con ruoli differenti sono coinvolti nello sviluppo di un progetto) e documentazione esterna rivolta all'utente, costituita ad esempio dal manuale d'uso del software prodotto.
Evidentemente ci saranno importanti corrispondenze fra documentazione interna ed esterna, anzi alcune parti saranno comuni ad entrambe: ad esempio le specifiche del prodotto e l'interfaccia devono essere conosciute sia dal programmatore che realizza l'applicazione, sia dal committente o dall'acquirente che la utilizzano.

2.2. ALGORITMI

PREREQUISITI
Saper Leggere – saper lavorare con metodo
OBIETTIVI
Saper cos'è un algoritmo strutturato

2.2.1. Cos'è un algoritmo

Si presentano alcune definizioni con lo scopo di evitare incomprensioni dovute ad ambiguità.

2.2.1.1. Algoritmo

Algoritmo è un **procedimento che può essere eseguito meccanicamente trasformando dei dati di input in risultati di output** e risulta essere:

- non ambiguo: univocamente interpretabile, non sono ammesse ad esempio espressioni del tipo "a piacere" oppure "abbastanza"
- deterministico: ad ogni passo dell'algoritmo deve essere determinato il passo successivo
- finito: composto di un numero finito di istruzioni, deve richiedere un numero finito di dati e deve terminare in un tempo finito, cioè deve avere anche in esecuzione un numero finito di passi
- eseguibile: ogni azione deve poter essere effettivamente eseguita (quando si scrive un algoritmo bisogna quindi tenere conto delle capacità dell'esecutore, devo utilizzare solo istruzioni elementari)
- generale: deve risolvere il problema per qualsiasi valore dei dati di ingresso (deve cioè risolvere una classe di problemi)
-

Il termine algoritmo deriva da **Al-Khowarizmi**, nome di un matematico persiano vissuto IX secolo, che definì le regole per eseguire le operazioni aritmetiche sui numeri scritti in notazione decimale (cioè le regole per il calcolo della somma, sottrazione, moltiplicazione e divisione fra numeri che ancora oggi tutti impariamo nelle scuole elementari).

Il concetto di algoritmo è precedente rispetto allo sviluppo dei calcolatori, si pensi che uno dei più antichi algoritmi, inteso come procedimento di calcolo per risolvere un problema, è l'algoritmo per il calcolo del Massimo Comun Divisore che Euclide nel suo libro (Elementi, 300 a.c.) riprendendolo da conoscenze precedenti.

Algoritmi equivalenti: per gli stessi dati di input si ottengono gli stessi risultati (le istruzioni di due algoritmi equivalenti possono quindi essere diverse in quanto si possono utilizzare diversi procedimenti per risolvere uno stesso problema).

Si pone allora il problema di scegliere il migliore fra due algoritmi equivalenti, cioè di individuare quale è più efficiente. Ma come possiamo valutare, cioè misurare , l'efficienza di un algoritmo?

L'*efficienza di un algoritmo* è determinata dalle risorse di tempo e spazio[32] richieste dall'algoritmo per essere eseguito.

È chiaro che un algoritmo deve cercare di ottenere il risultato richiesto nel minor tempo possibile ed utilizzando il minor numero di risorse. Attenzione però: a seconda dei dati di ingresso ci possono essere situazioni ottimali in cui l'algoritmo risulta essere molto efficiente.
In genere dovremo valutare l'efficienza dell'algoritmo confrontando i risultati ottenuti nel caso migliore, nel caso medio e nel caso peggiore.
Un esempio: un bibliotecario deve sistemare i libri in un certo palchetto in ordine alfabetico di autore. Può seguire diversi metodi di lavoro, e cercare di capire, per la prossima volta che dovrà eseguire un compito analogo, qual è il più efficiente. Dovrà valutare diversi casi: il migliore si verifica quando, per caso appunto, i libri risultano già in ordine! Ancora: le risorse richieste sono correlate al numero dei dati da trattare, nel caso della biblioteca sarà diverso dover riordinare un palchetto o un intero scaffale.

2.2.1.2. Algoritmo strutturato

Abbiamo provato a costruire la tabella di traccia dell'algoritmo di Euclide proposto nella forma riportata a pagina 30 . È chiaro che per affrontare problemi più complessi è necessario scrivere i nostri algoritmi in modo che non diventino troppo complicati, in particolare evitando istruzioni del tipo "vai alla riga 1" o "vai alla riga 5"; prova ad immaginare cosa potrebbe succedere con molte istruzioni: semplicemente si perderebbe il controllo della logica del nostro algoritmo, con il rischio di introdurre inavvertitamente errori di vario genere.

É necessario scrivere algoritmi di buona qualità: **Niklaus Wirth**[33] nel 1968 ha definito i principi delle **programmazione strutturata**[34]. L'idea è quella di dare una disciplina nella realizzazione degli algoritmi, specificando le istruzioni e le strutture che possono essere utilizzate:

- Istruzioni semplici
  - Input/ Output
  - Assegnazione
- Strutture di controllo
  - sequenza
  - selezione
  - ripetizione

Tutti gli algoritmi che si propongono in questo testo sono strutturati.

---

[32] Il tempo è determinato dal numero di operazioni da effettuare, lo spazio dal numero di variabili utilizzate

[33] Wirth (nato nel 1934) ha progettato il linguaggio di programmazione Pascal ed è stato neglianni 70 del secolo scorso uno dei fautori del metodo di sviluppo TOP- DOWN; autore di testi significativi fra cui ricordiamo "Algoritmi + Strutture Dati = Programmi" e "Principi di Programmazione Strutturata".

[34] Un programma è l' implementazione dell' algoritmo , coè la traduzione di un algoritmo in un qualsiasi linguaggio di programmazione, tale che possa in seguito essere eseguito da un computer.

2.2.1.3. Teorema di Böhm - Jacopini

Per scrivere un buon algoritmo è necessario verificare che esso sia strutturato, ma sorge un problema: potrebbero esserci problemi particolari per risolvere i quali si renda necessario un algoritmo non strutturato? Questa potrebbe essere una limitazione importante!

Böhm - Jacopini nel 1966 enunciano[35] e dimostrano un teorema che afferma che un algoritmo non strutturato può sempre essere trasformato in uno strutturato equivalente. Questo è il teorema che ha dato forza negli anni successivi ai fautori dello sviluppo degli algoritmi strutturati.

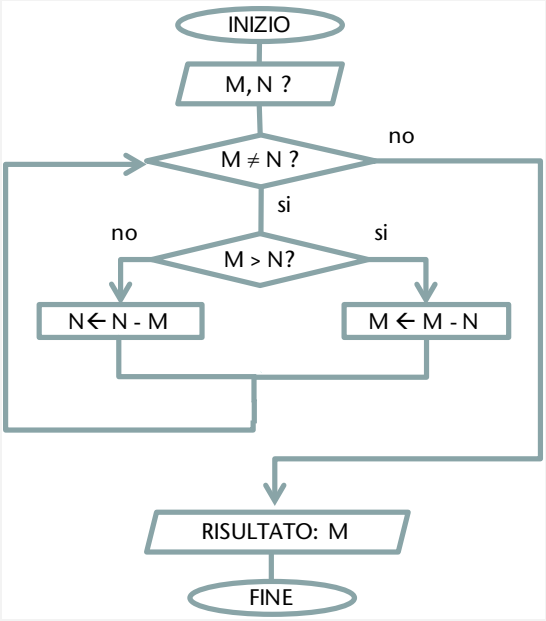
2.2.1.4. Rappresentazione degli algoritmi

Un algoritmo è la sequenza di operazioni elementari per risolvere un problema: come è possibile descriverle? Di norma non si utilizza il linguaggio naturale, che rischia di introdurre ambiguità, ma rappresentazioni più formali.
Si propone ancora l'algoritmo per il calcolo del massimo comun divisore, naturalmente ora nella sua versione strutturata, mostrandolo nelle rappresentazioni più diffuse.

Diagramma a Blocchi (DaB)

attualmente ripreso nello standard UML (diagramma di attività)

rappresentazione grafica, ogni istruzione semplice ed ogni struttura è rappresentata da un blocco



[35] Corrado Böhm - Giuseppe Jacopini, "Flow diagrams, turing machines and languages with only two formation rules", Communications of the ACM, Volume), Numero 5, maggio 1996, pp 366-371

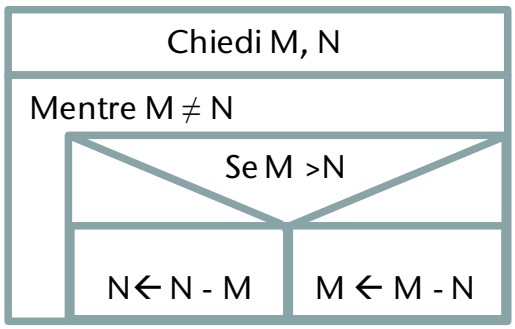
Notazione lineare strutturata (NLS) o pseudo-codice

utilizza costrutti linguistici non ambigui, simile ad un linguaggio di programmazione

```
inizio
    chiedi M, N
    mentre ( M ≠ N)
        se ( M > N) allora
            M ← M - N
        altrimenti
            N ← N - M
fine
```

Grafo di Nassi-Schneidermann

rappresentazione grafica fortemente orientata a mettere in evidenza le strutture



2.3. DIAGRAMMI A BLOCCHI STRUTTURATI

PREREQUISITI

Saper Leggere – saper lavorare con metodo

OBIETTIVI

Saper scrivere algoritmi strutturati per la soluzione di problemi semplici

2.3.1. Rappresentazione degli algoritmi mediante diagrammi a blocchi strutturati

Rappresentazione degli algoritmi mediante diagrammi a blocchi strutturati

2.3.1.1. Inizio/fine

I diagrammi a blocchi utilizzano una rappresentazione grafica in cui ogni forma geometrica ha un significato ben preciso: ciò guida nella lettura del diagramma stesso, che può essere effettuata a due livelli: in forma sintetica si pone l'attenzione solo sulle strutture individuate proprio dalla forme geometriche, in forma più analitica si considerano anche tutti i dettagli.

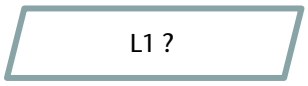
Ogni algoritmo strutturato presenta un blocco di inizio ed un blocco di fine.

2.3.1.2. Input/output

Un algoritmo deve risolvere non un singolo problema, bensì una classe di problemi, cioè tutti quei problemi in cui la logica della soluzione è la medesima, ma che si differenziano fra loro per il valore delle variabili.

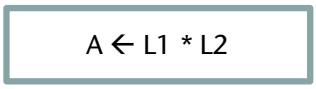
Ad esempio uno dei problemi esaminati (vedi 2.1.3.2 e 2.1.3.3) si riferisce al calcolo di perimetro ed area di un qualsiasi rettangolo del quale si conoscano le misure dei lati: è chiaro che ci deve essere un modo per indicare di volta in volta, durante l'esecuzione, i valori specifici che ci interessano (dati di input).
In modo analogo si deve prevedere di poter comunicare all'utilizzatore i risultati ottenuti (output).

L'Input e **l'output** vengono rappresentati graficamente mediante parallelogrammi che contengono il nome della variabile cui ci si riferisce.
In questo testo il punto interrogativo indica l'input (sto chiedendo che l'utente specifichi il valore), l'output mostra invece l'indicazione che le variabili in questione rappresentano un risultato.



2.3.1.3. Assegnazione

Esaminiamo la seguente istruzione:



essa deve essere interpretata come segue:

L1 * L2 calcola il valore dell'espressione indicata a destra (Right value), in questo caso esegue la moltiplicazione

A ← assegna, cioè attribuisci, il valore calcolato in precedenza alla variabile A (Left Value)

Un altro esempio:

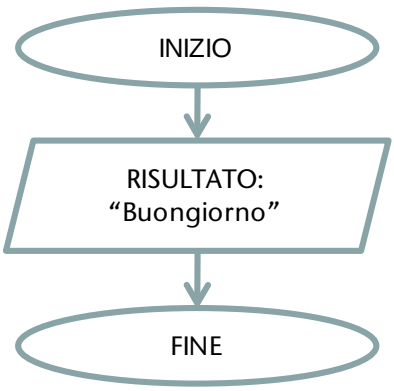
Y← Y+ 1

Qui è chiaro il significato di assegnazione. In matematica non posso dire che y=y+1, noi però stiamo utilizzando l'operazione di assegnazione, ciò significa che se ad esempio la variabile Y ha il valore 5 in seguito a questa operazione il suo valore verrà modificato passando a 6.

2.3.1.4. Connettori

I connettori sono uno dei componenti fondamentali nella realizzazione di un diagramma a blocchi, in quanto servono ad indicare i collegamenti fra di essi.

In un diagramma strutturato ogni blocco, che sia semplice come quelli visti fin qui, o che sia strutturato come quelli mostrati nei prossimi paragrafi, ha sempre uno ed un solo punto di ingresso ed uno ed un solo punto di uscita (tranne naturalmente i blocchi di inizio e fine) : i connettori collegano i vari blocchi attraverso questi punti (il punto di uscita da un blocco verrà collegato con il punto di ingresso del blocco successivo).



2.3.2. Sequenza

2.3.2.1. Blocco sequenza

Consideriamo un semplice problema: si vogliono calcolare il perimetro e l'area di un rettangolo.

Analisi

Dovremo conoscere le misure dei due lati quindi, ricordando le opportune formule di geometria, potremo calcolare i valori richiesti.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
L1	Primo lato del rettangolo	Input
L2	Secondo lato del rettangolo	Input
P	Perimetro	Output
A	Area	Output

Algoritmo

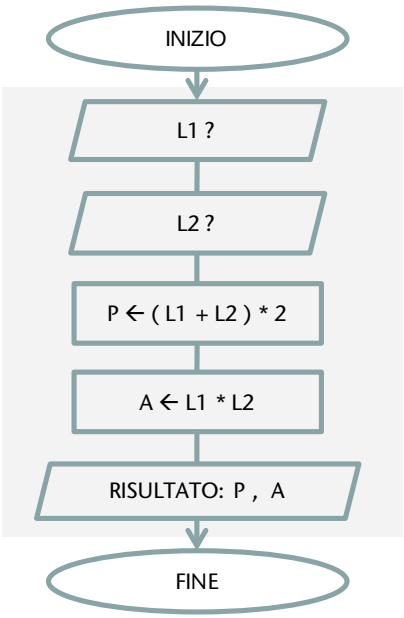


Tabella di traccia

L1	L2	P	A	Output
5				
	10			
		30		
			50	
				30, 50

Nell'algoritmo dell'esempio tutte le istruzioni sono eseguite una di seguito all'altra: si è utilizzato un **blocco sequenza**.

Problema→ si vuole calcolare il doppio di un certo valore dato.

Analisi

Dovremo conoscere il valore iniziale, quindi calcolare il doppio e comunicare il risultato ottenuto.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
V	Valore iniziale, che poi viene raddoppiato	Input/Output

Algoritmo

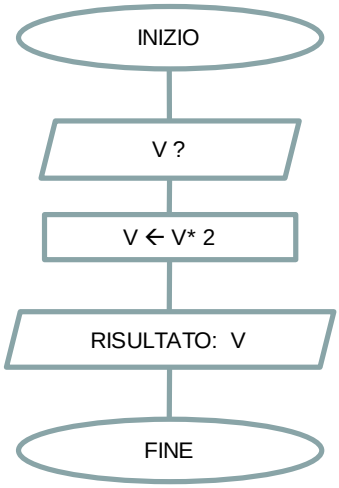


Tabella di traccia

V	Output
7	
14	
	14

Abbiamo visto un altro esempio di assegnazione: si considera il valore iniziale della variabile V (in questo esempio 7), si calcola V*2, ottenendo 14, di seguito si assegna il valore calcolato ancora alla variabile V, che quindi assume il nuovo valore 14.

2.3.3. Selezione

2.3.3.1. Blocco selezione: Se ... Altrimenti

Problema→ Conoscendo il voto ottenuto da un alunno si vuole inviare un messaggio adeguato: congratulazioni in caso di valutazione positiva, altrimenti un invito ad una maggiore applicazione nello studio.

Analisi
Chiediamo subito di conoscere il voto ottenuto, al fine di personalizzare il messaggio in modo adeguato.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
V	Voto ottenuto dallo studente	Input

Algoritmo

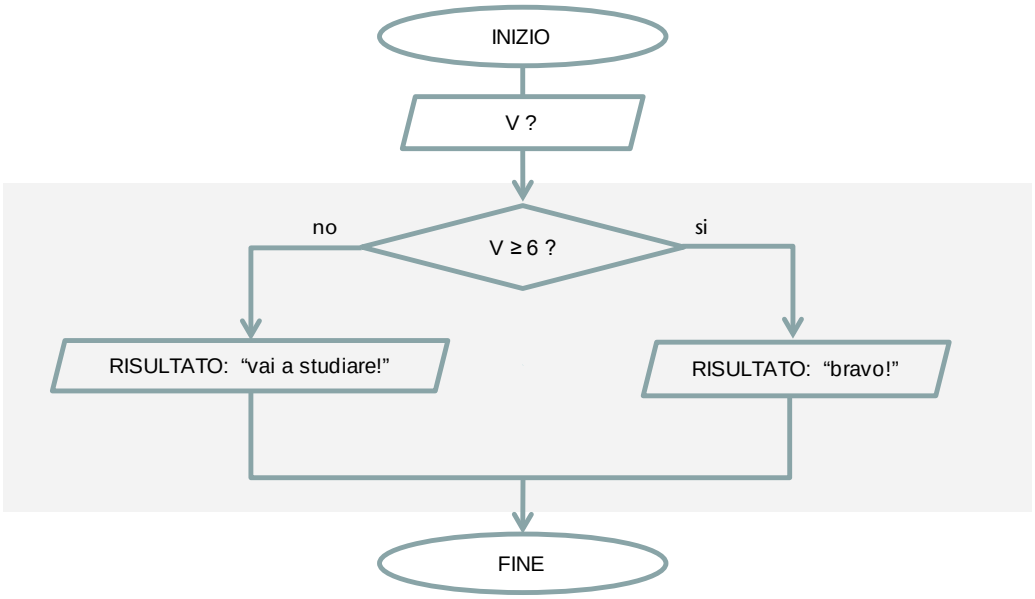


Tabella di traccia (caso 1)

V	Output
7	
	Bravo!

Tabella di traccia (caso 2)

V	Output
4	
	Vai a studiare!

Si osserva che la **condizione** indicata nel rombo deve essere tale che la risposta possa essere solo vero oppure falso (o se si preferisce si/no), senza consentire alcuna ambiguità né alcun dubbio.

2.3.3.2. Blocco selezione: Se

Problema→ Si stabilisce di assegnare un premio a tutti gli alunni che hanno ottenuto un punteggio di 10 in una determinata prova: si richiede, conoscendo il voto di un alunno, di specificare l'eventuale premio ottenuto.

Analisi
Chiediamo subito di conoscere il voto ottenuto, in modo da informare, se il voto è pari a 10, della vincita del premio; in caso contrario non viene inviato alcun messaggio.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
V	Voto ottenuto dallo studente	Input

Algoritmo

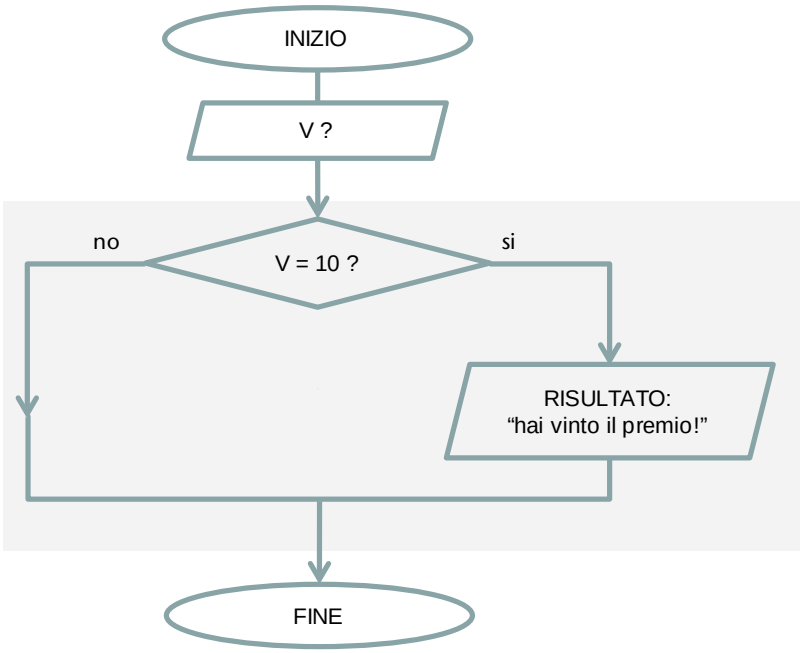


Tabella di traccia

V	Output
7	
	Hai vinto il premio!

Questo è un caso particolare del precedente: nel blocco selezione deve sempre essere presente il ramo che indica le azioni da svolgere quando la condizione è verificata, possono eventualmente anche essere specificate le azioni da compiere, in alternativa, nel caso negativo.

2.3.3.3. Blocchi selezione annidati

Problema→ Conoscendo il voto ottenuto da un alunno si vuole inviare un messaggio adeguato: in caso di valutazione positiva congratulazioni e, con votazione pari o superiore a 9, annuncio della vincita di un premio, altrimenti un invito ad una maggiore applicazione.

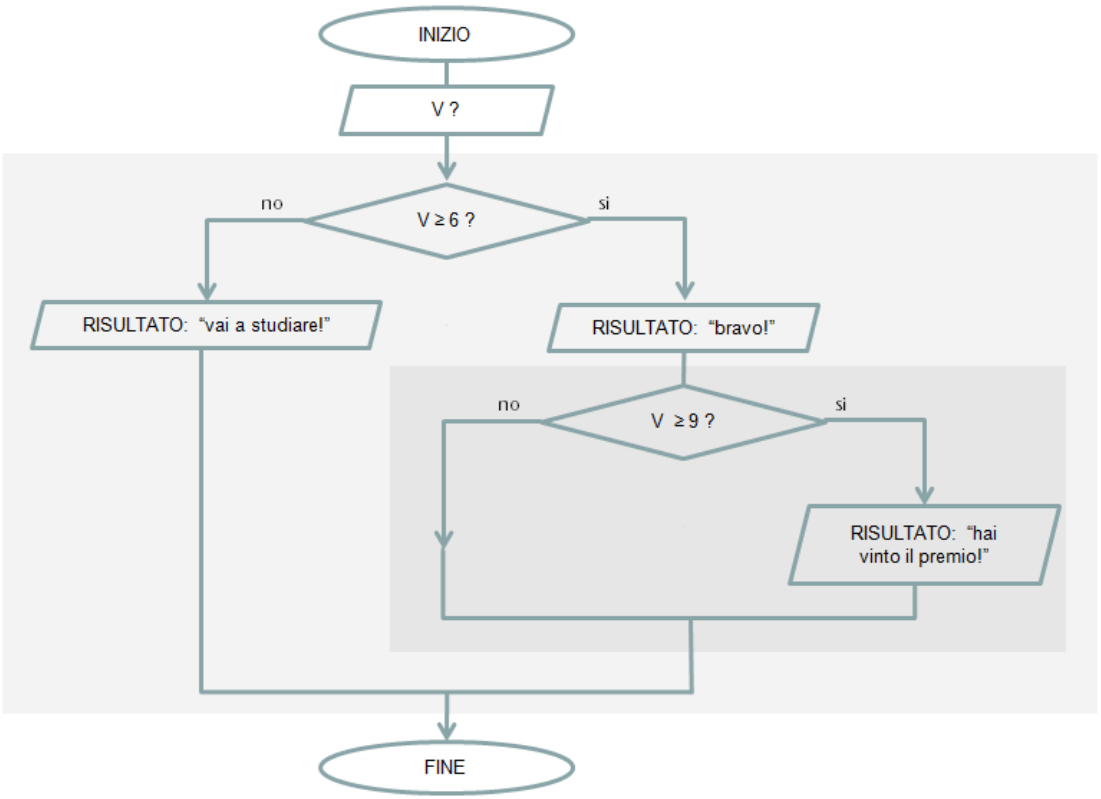
Analisi

Chiediamo subito di conoscere il voto ottenuto per poter personalizzare il messaggio in modo adeguato: da osservare che nel caso di valutazione positiva non è sufficiente limitarsi alle congratulazioni ma, eventualmente, bisogna anche informare circa il premio vinto.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
V	Voto ottenuto dallo studente	Input

Algoritmo



Blocco annidato significa che un blocco viene inserito all'interno di un altro (opposto di blocchi in sequenza, cioè uno di seguito all'altro).

Tabella di traccia (caso 1)

V	Output
5	
	Vai a studiare!

Tabella di traccia (caso 2)

V	Output
7	
	Bravo!

Tabella di traccia (caso 3)

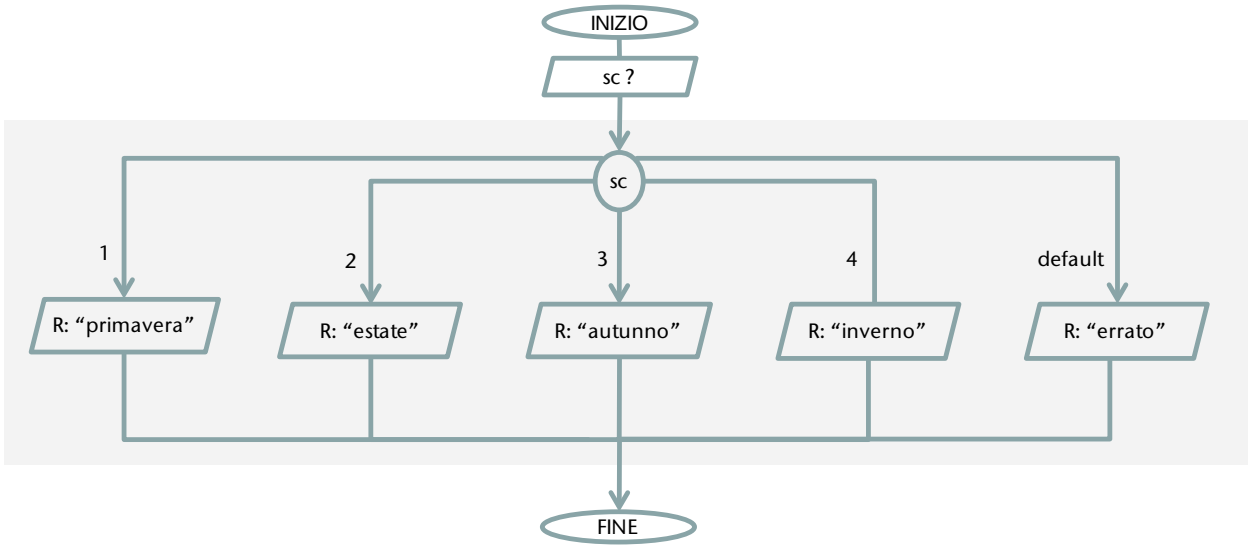
V	Output
9	
	Bravo!
	Hai vinto il premio!

2.3.3.4. Blocco selezione multipla

Problema→ L'utente specifica mediante un numero la stagione che preferisce (1 per primavera, 2 per estate, e così via): si richiede di visualizzare il nome corrispondente alla stagione selezionata.

Questo problema viene proposto in quanto ci consente alcune riflessioni. Si potrebbe realizzare l'algoritmo corrispondente utilizzando una serie di blocchi selezione, tutti simili fra loro, ma pensiamo: e se ci venisse chiesto di gestire non 4 stagioni, ma ad esempio 12 mesi? Ancora tanti (troppi!) blocchi selezione.
Si presenta qui una struttura che, seppure non indispensabile[36], pur tuttavia agevola la gestione di situazioni simili a quella presentata.
Il blocco selezione ha due soli rami (vero e falso), qui sostituiamo il rombo della condizione con un cerchio, così possiamo collegare molti rami, tanti quanti ci servono, ciascuno corrispondente ad un determinato valore della variabile di controllo (in questo caso sc, rappresenta la stagione specificata dall'utente) ed otteniamo quanto richiesto (R è l'abbreviazione di Risultato, indica il valore di output).

Algoritmo



Si osservi che se viene indicato un valore diverso da quelli previsti, e quindi in mancanza (default) di indicazioni corrette, è possibile comunque gestire l' errore riscontrato.
Codifica: vedi paragrafo Istruzione switch

[36] Ricorda ricorda infatti Wirth (vedi paragrafo 2.2.1.2) e Böhm - Jacopini (vedi paragrafo 2.2.1.3)

2.3.4. Ripetizione

2.3.4.1. Blocco Ripetizione (controllo in testa – condizione iniziale)

Problema→ Elencare, su richiesta dell'utente, i numeri 1, 2, 3, 4 .... e così via

Analisi

Chiediamo subito all'utente di confermare la richiesta di ricevere un numero dell'elenco e quindi lo visualizziamo, continuando sempre fino a quando viene confermata la richiesta del nuovo numero.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
Elenco	Memorizza la risposta dell'utente alla richiesta di conoscere un numero dell'elenco	Input
I	Valore da visualizzare	Output

Algoritmo

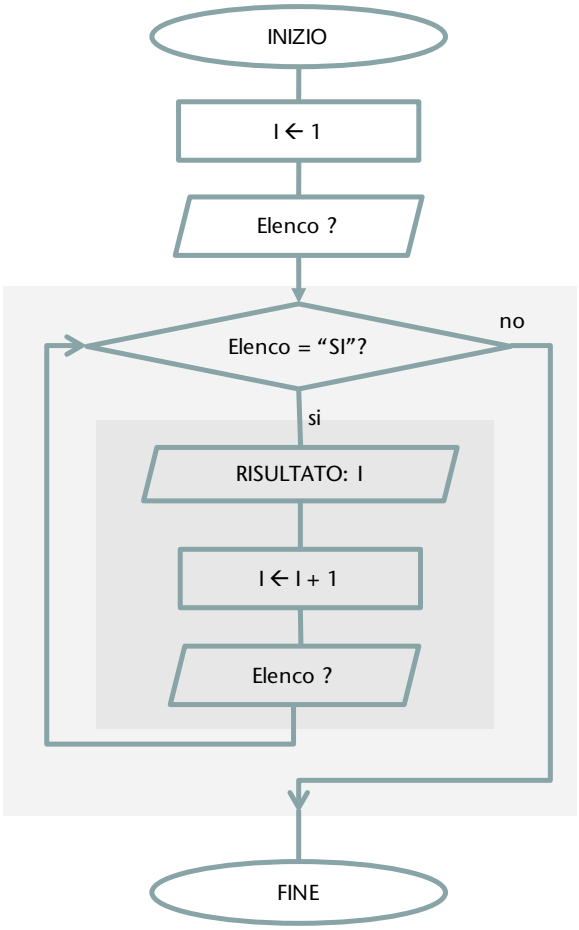


Tabella di traccia

Elenco	I	Output
	1	
si		
		1
si		
	2	
		2
si		
	3	
		3
no		
	4	

Questo costrutto è molto importante: ogni volta che ci si trova davanti ad una situazione ripetitiva, cioè ad operazioni che vanno eseguite sempre nello stesso modo, invece di utilizzare la struttura della sequenza, e scrivere ripetutamene le istruzioni in questione, si utilizza il blocco ripetizione.

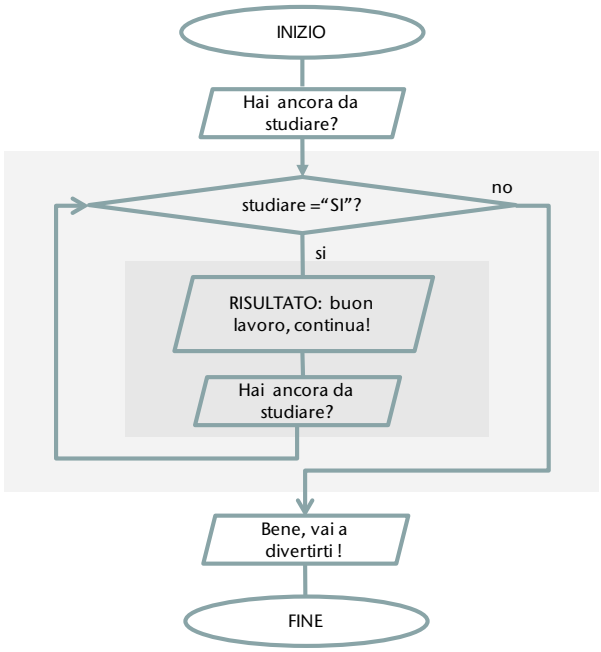
Le **istruzioni** da ripetere, che sono **scritte una sola volta nel corpo del blocco ripetizione**, rappresentato in figura con uno sfondo più scuro, **durante l'esecuzione vengono ripetute** tutte le volte che è necessario, e precisamente tutte le volte che viene verificata, cioè che risulta vera, la condizione posta all'inizio.

Quando la condizione risulta falsa si esce dal blocco ripetizione.

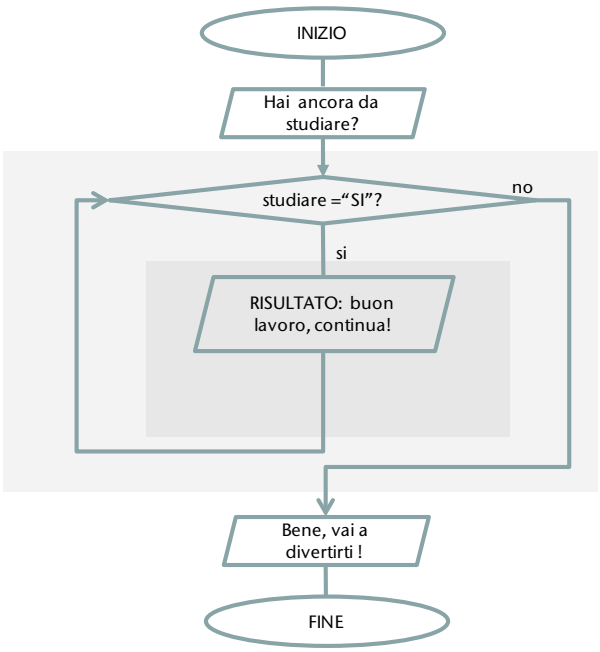
Un caso particolare significativo è quello in cui la condizione non sia mai verificata: le istruzioni del ciclo non vengono eseguite neanche una volta.

Nel blocco da ripetere deve essere prevista almeno una istruzione che possa modificare la condizione iniziale, in modo da controllare l'uscita.

Prova a dire qual è il risultato prodotto dai due seguenti algoritmi ed esprimi una tua opinione: qual è quello giusto?



Al termine degli studi puoi andare a divertirti.



Qui si ha un **loop infinito**, cioè si ripetono sempre le operazioni indicate nel ciclo, senza mai uscirne in quanto, una volta entrati nel ciclo, non ci si chiede si è arrivato il momento di concludere lo studio, quindi . . . non si interrompe mai!

2.3.4.2. Blocco Ripetizione (con contatore)

Problema→ Scrivere i numeri da 1 a N .

Analisi

Chiediamo subito di conoscere il valore di N, che rappresenta l'ultimo dei numeri da visualizzare, quindi procediamo, uno per uno, a mostrare tutti i valori richiesti.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
N	Valore finale da visualizzare	Input
I	Contatore, tiene traccia di quante volte si sono eseguite le istruzioni del ciclo	Output

Algoritmo

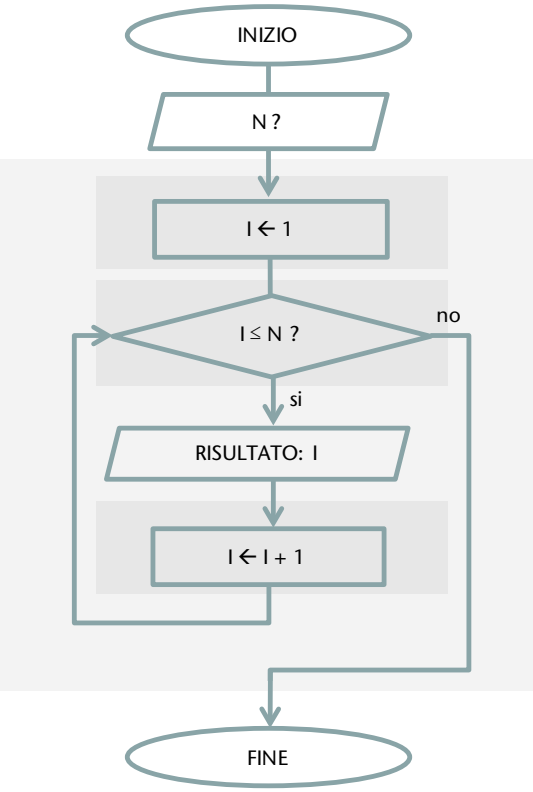


Tabella di traccia

N	I	Output
4		
	1	
		1
	2	
		2
	3	
		3
	4	
		4
	5	

Questo è un caso particolare, ma molto significativo, del blocco ripetizione visto nel paragrafo precedente: valgono tutte le regole generali (blocco di istruzioni che si ripetono, controllate dal verificarsi di una condizione iniziale) ed inoltre si conosce a priori, cioè prima di entrare nel ciclo, quante volte si vogliono eseguire le istruzioni da ripetere: è sufficiente quindi utilizzare un contatore come variabile di controllo dell'esecuzione del ciclo.

2.3.4.3. Blocco Ripetizione (controllo in coda – condizione finale)

Problema→ Scrivere il numero 1, quindi di seguito i numeri 2, 3, 4 .... fino a quando l'utente chiede di continuare l'elenco.

Analisi

Per risolvere il problema dobbiamo subito scrivere il numero 1, come ci viene richiesto, quindi dare la possibilità di continuare eventualmente l'elenco: a tal proposito dopo la stampa di un qualsiasi numero chiediamo all'utente il suo interesse a conoscere il numero successivo , continuando sempre fino a quando si riceve la conferma.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output
Continuo	Memorizza la risposta dell'utente alla richiesta di continuare l'elenco di numeri	Input
I	Valore da visualizzare	Output

Algoritmo

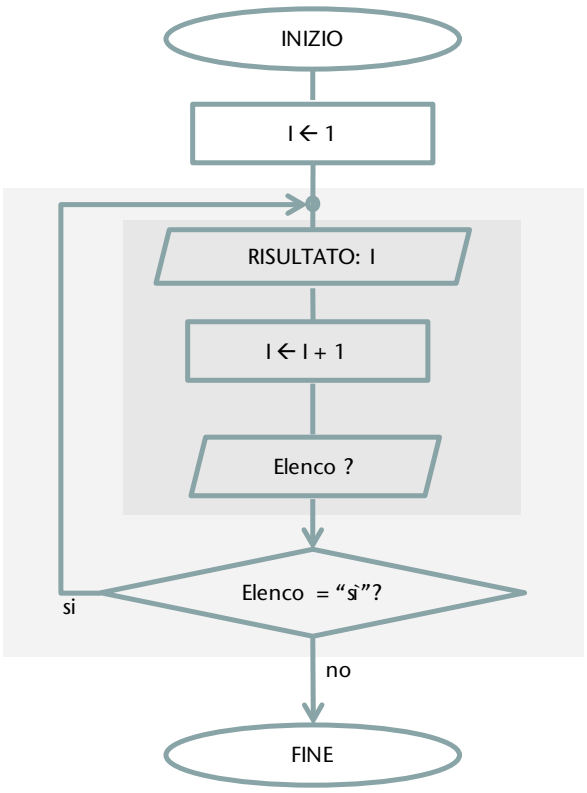


Tabella di traccia

Elenco	I	Output
	1	
		1
	2	
si		
		2
	3	
si		
		3
no		

Anche questo è un caso particolare e significativo di blocco ripetizione: possiamo utilizzare questo costrutto quando siamo sicuri di dover eseguire almeno una volta le istruzioni del ciclo (al contrario della situazione discussa nel paragrafo 2.3.4.1 Blocco Ripetizione (controllo in testa – condizione iniziale)).

A conclusione di questa unità riflettiamo sul fatto che gli esempi proposti in effetti non costituiscono problemi reali, per il momento si è trattato quasi solo di esercizi. Però pensiamo ad esempio a quest'ultima proposta: una richiesta reale potrebbe essere quella di conoscere sempre dei numeri uno dopo l'altro, ma fornendo valori casuali. Ancora: la successione di questi valori casuali potrebbe essere la base per implementare una versione elettronica del gioco della tombola. Questo indica l'importanza di assimilare i principi di base, per poter poi riutilizzare in contesti diversi le conoscenze e competenze acquisite.

2.3.5. Soluzione di problemi semplici

Il procedimento risolutivo di un problema deve sempre essere individuato dall'uomo e costituisce uno degli aspetti del lavoro di chi si occupa di informatica: fondamentale utilizzare a tale scopo la tecnica del problem solving. Viene qui presentata la soluzione di alcuni problemi semplici.

2.3.5.1. Scambio di due valori

Problema→ Per scherzo è stato messo il caffè nel calice dello spumante, e viceversa. È necessario scambiarli.

Le figure mostrano cosa fare per risolvere il problema, naturalmente con l' aiuto di un terzo contenitore, il bicchiere in questo caso.

calice con caffè

tazzina con spumante

serve un bicchiere per aiutarci a scambiare il contenuto del calice e quello della tazzina

calice vuoto, il caffè è stato messo nel bicchiere

calice con spumante, OK!

tazzina vuota, lo spumante è stato messo nel calice

tazzina con il caffè, OK!

Problema→ Scambiare i valori di due variabili .

Per risolvere il problema si utilizzerà una strategia del tutto analoga a quella del caso precedente.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output / lavoro
V1	Primo Valore specificato dall'utente[37]	Input / Output
V2	Secondo Valore specificato dall'utente	Input
C	variabile di aiuto	lavoro

Algoritmo

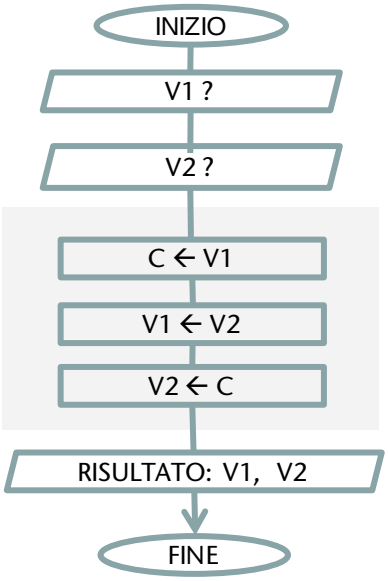


Tabella di traccia

V1	V2	C	Output
7			
	2		
		7	
2			
	7		
			2, 7

[37] L' utente è colui che utilizzerà l' applicazione ottenuta da questo algoritmo, ora utente è l' esecutore dell' algorimo, cioè chi lo usa per risolvere il problema.

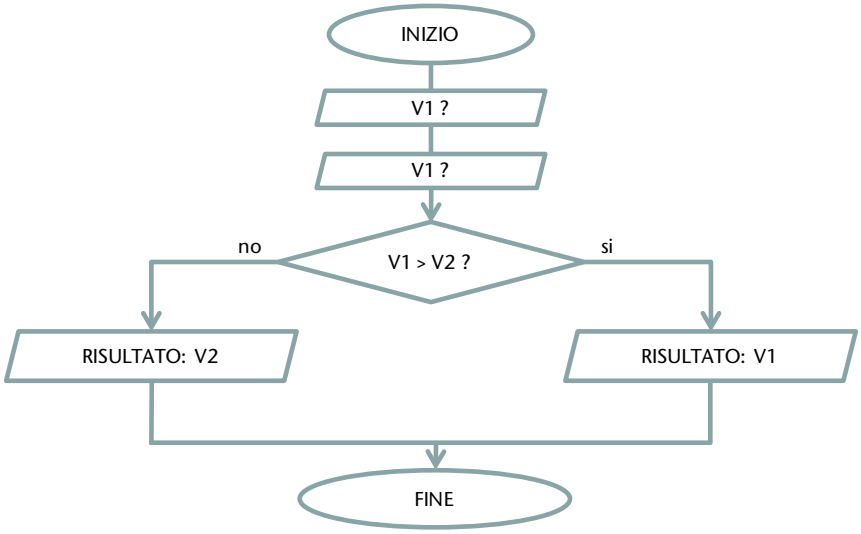
2.3.5.2. Massimo di due valori

Problema→ Dati due valori, calcolare il massimo.

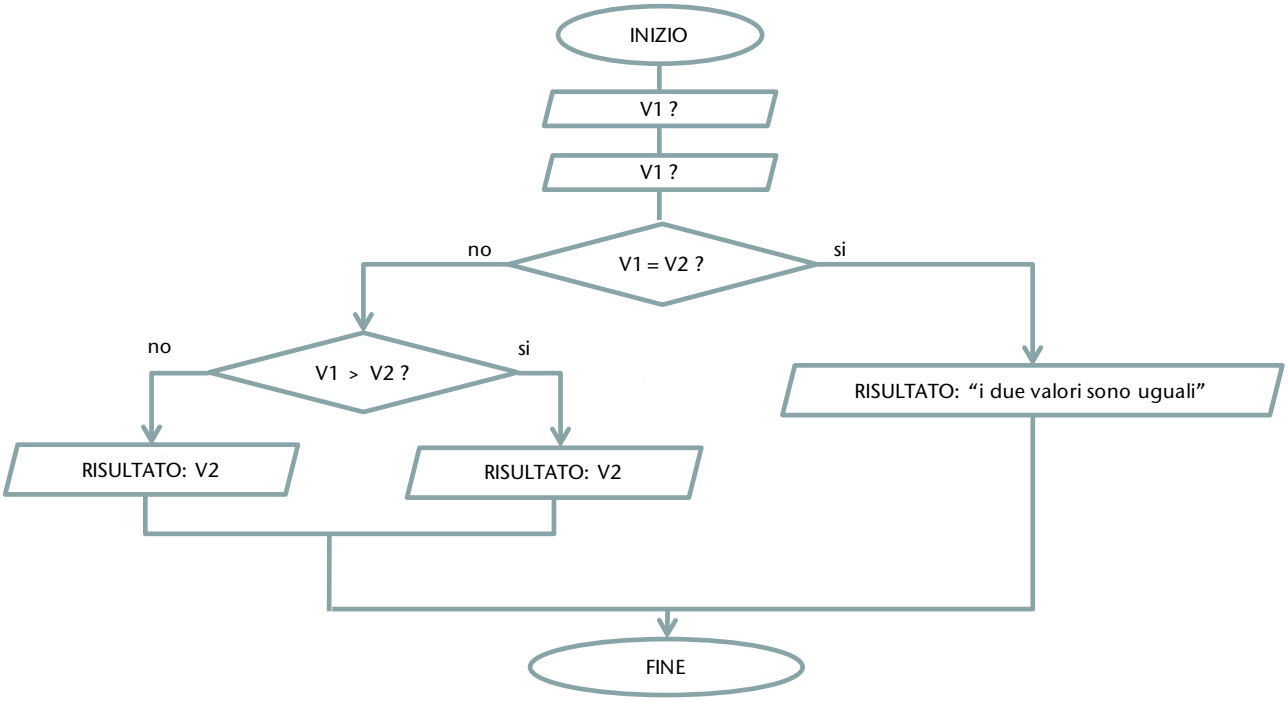
Analisi

Chiediamo subito di conoscere i due valori, che possiamo memorizzare nelle variabili V1 e V2, quindi con un semplice confronto selezioniamo il valore da presentare in output.
Una analisi più approfondita ci porta però a considerare che non necessariamente un valore è maggiore dell'altro, in quanto essi possono risultare uguali. Proviamo a considerare anche questo caso particolare.

Algoritmo (1^ versione)



Algoritmo (2^ versione)



2.3.5.3. Contatori e accumulatori

Problema→ Calcolare la media di 3 valori.

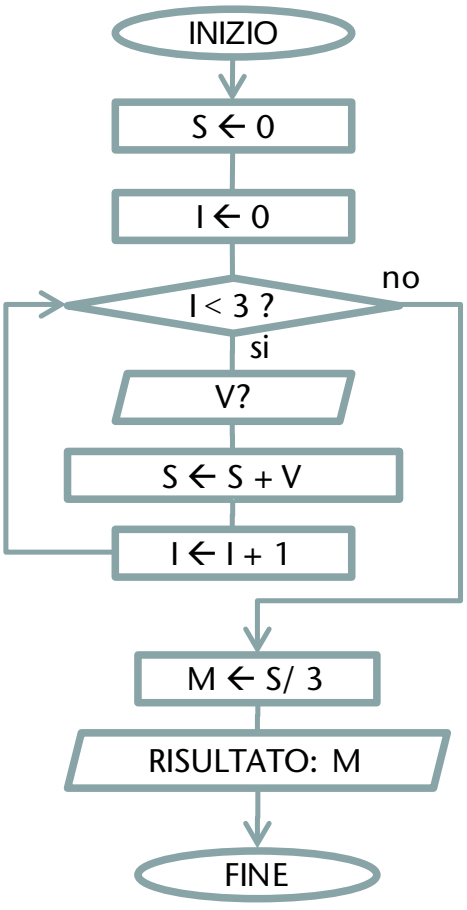
Analisi

Il problema è semplice, ma cerchiamo di generalizzarlo, prendendo in considerazione un qualsiasi numero di valori, e risolvendo il caso particolare in cui il numero dei valori da considerare sia proprio 3.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output/ lav.
V	Valore specificato dall'utente I diversi valori vengono memorizzati uno per volta	Input
I	**contatore**: serve per contare quanti valori sono stati dati dall'utente Verrà inizializzato a zero, perché all'inizio l'utente non ha ancora fornito nessun valore, poi man mano viene incrementato di uno in corrispondenza di ogni valore dato	lavoro
S	**accumulatore**: serve per calcolare la somma dei valori dati, cioè per accumulare le quantità man mano che i valori sono specificati dall'utente	lavoro
media	media dei valori dati	Output

Algoritmo



Codifica: vedi paragrafi 3.3.2.3 e 3.3.2.4.

Tabella di traccia

I	V	S	Media	Output
0		0		
	7			
		7		
1				
	30			
		37		
2				
	4			
		41		
3				
			13.6	
				13.6

2.3.5.4. Massimo di N valori

Problema→ Dato un certo numero di valori calcolare il massimo

Analisi

Innanzi tutto devo sapere quanti sono i valori da considerare. Quindi chiedo di conoscere il primo valore: in questo momento tale valore è senz'altro il massimo. Chiedo di conoscere il valore successivo: se è più grande del massimo provvisorio lo devo ricordare: al momento è proprio questo il nuovo valore massimo, sempre provvisorio (poi è da vedere, dipende da come saranno gli altri valori, quelli che non conosco ancora, proprio come succede ad esempio in una gara a cronometro). Ripeto questo ragionamento per ogni valore: al termine posso affermare che quello che consideravo massimo provvisorio è a tutti gli effetti il massimo, in quanto non ci sono più altri valori che lo possano superare.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output/lavoro
N	Numero totale di valori da considerare	Input
Max	valore Massimo	Output
I	contatore dei valori che sono stati richiesti	Output

Algoritmo

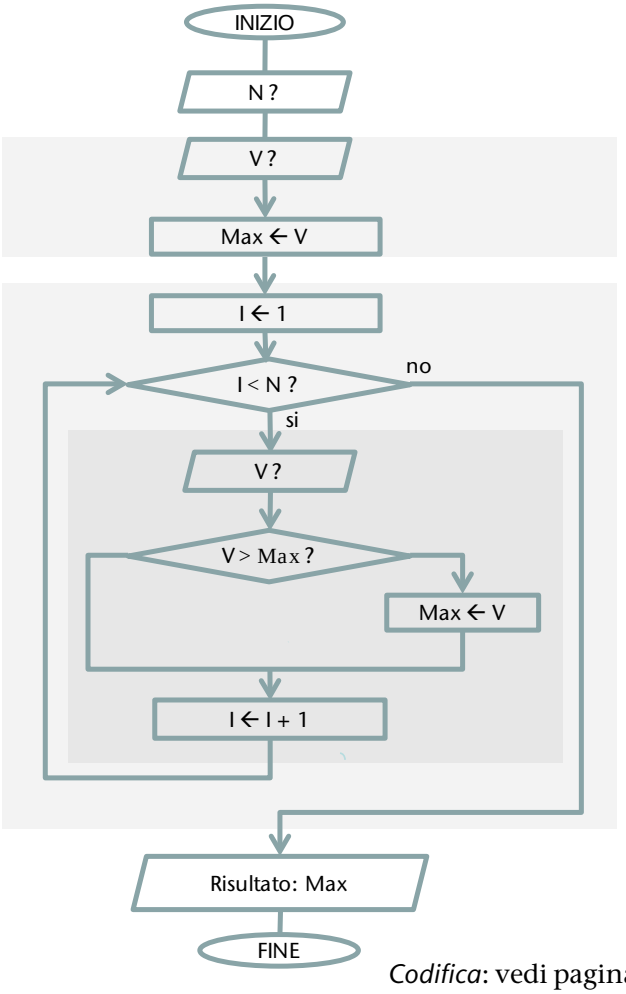


Tabella di traccia

N	V	I	Max	Output
4				
	5			
			5	
		1		
	4			
		2		
	7			
			7	
		3		
	6			
		4		
				7

Codifica: vedi pagina 88

2.3.5.5. Carrello della spesa

Problema→ Calcolare il totale da pagare per acquitare i prodotti contenuti in un carrello della spesa.

Analisi
Quando ci si presenta alla cassa si è acquistato almeno un prodotto, quindi si comincia con chiedere di specificare il costo corrispondente, poi si seguito si indicano i costi di tutti i successivi prodotti: il termine della lista può essere indicato dal valore zero (che sicuramente non rappresenta il costo di alcun prodotto). Di volta in volta naturalmente si aggiorna il totale dei prodotti acquistati.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output/lavoro
Costo	costo di un singolo prodotto della lista	Input
Totale	Totale da pagare per i prodotti acquistati	Output

Algoritmo

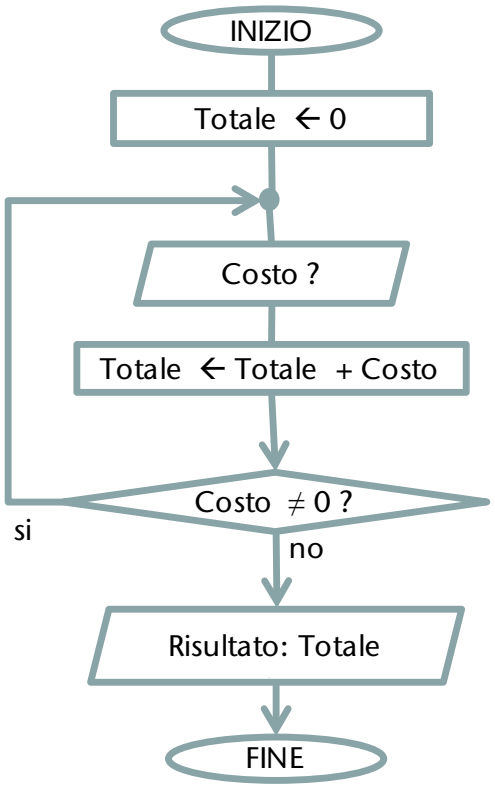


Tabella di traccia

Costo	Totale	Output
	0	
12,5		
	12,50	
3		
	15,50	
5,6		
	21,10	
5,35		
	26,45	
0		
		26,45

Codifica: vedi pagina 86.

2.3.5.6. Gestione menu

Problema→ Realizzare una piccola calcoltrice che effettui le 4 operazioni forndamenli fra due numeri scelti dall' utente.

Analisi

Si presenta all' utente un menu che gli permette di scegliere fra diverse opzioni:

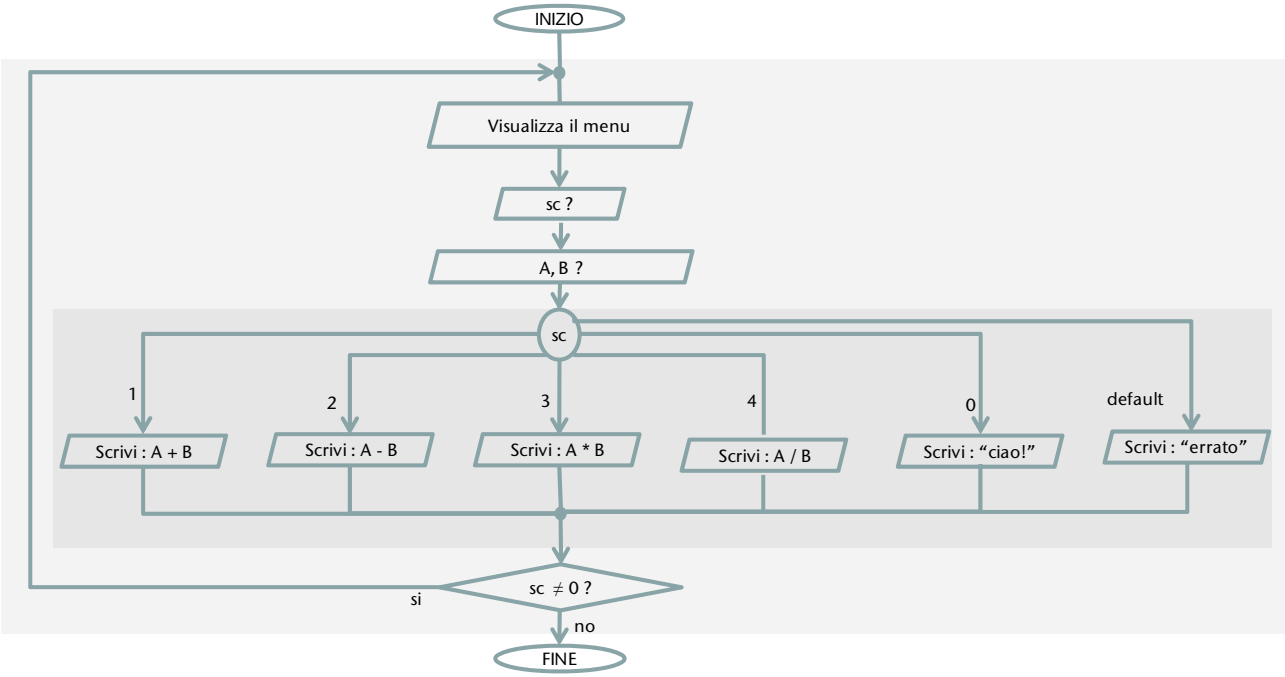
1. inserimento valori degli operandi
2. somma
3. sottrazione
4. moltiplicazione
5. divisione
0. fine

A seguito della scelta operata dall'utente la nostra applicazione risponde in modo conveniente eseguendo l'attività richiesta, quindi visualizza nuovamente il menu, finché l' utente non decide di uscire.

Tabella di descrizione delle variabili

Nome	Descrizione	Input/Output/lavoro
A	primo operando	Input
B	secondo operando	Input
R	risultato operazione	Output
sc	opzione scelta dal menu	Input

Algoritmo



Codifica: vedi pagina 89.

HAI IMPARATO A ...

1. Risolvere probemi semplici con un buon metodo di lavoro
2. Riconoscere i dati ed i risultati
3. Costruire algoritmi strutturati
4. Verificare la correttezza del tuo lavoro

3. Introduzione alla programmazione: linguaggio C#

In questa sezione si introduce la programmazione in linguaggio C# come mezzo per la risoluzione di problemi con l'uso del computer .

- Linguaggi di programmazione
- Ambiente di sviluppo Microsoft Visual Studio
- Iniziamo a programmare

3.1. LINGUAGGI DI PROGRAMMAZIONE

PREREQUISITI

Conoscere gli algoritmi

OBIETTIVI

Saper scrivere un programma

3.1.1. Tipologie di linguaggi

3.1.1.1. Linguaggi naturali ed artificiali

Quando devo parlare con qualcuno in genere mi esprimo in italiano, ma se il mio interlocutore è straniero spesso utilizzo un'altra lingua: in generale si tratta di trovare una forma di comunicazione efficace, cioè tale che permetta a ciascuno di esprimersi a di comprendere l'altro. Posso *parlare* anche al mio cane: in questo caso si stabilisce comunque una forma di comunicazione, anche non verbale, che si basa su segnali e simboli il cui significato è condiviso.

Possiamo effettuare una distinzione

- **linguaggio naturale** : può presentare un certo livello di ambiguità
- **linguaggio artificiale**: è definito in modo non ambiguo

Il linguaggio è uno strumento che serve per comunicare: se in particolare mi interessa che un elaboratore possa eseguire delle operazione devo trovare il modo di comunicare le mie richieste, cioè devo pormi il problema di avere un linguaggio in comune. Siccome il calcolatore esegue automaticamente le operazioni richieste è indispensabile che il ingaggio che utilizziamo non presenti alcuna ambiguità che il computer, essendo una macchina, non saprebbe come interpretare: sarà quindi necessario comunicare con un linguaggio artificiale.

3.1.1.2. Programmazione: linguaggio macchina e linguaggi artificiali di alto livello

Fra i linguaggi artificiali con cui comunichiamo con il computer possiamo distinguere diverse tipologie; ne riportiamo alcune in elenco:

- **linguaggi di programmazione**
  - **linguaggio macchina (binario)** in cui le comunicazioni avvengono direttamente con l'hardware, le istruzioni da impartire al computer sono espresse utilizzando solo zero (circuito aperto) e uno (circuito chiuso) -parliamo quindi di linguaggio binario- e sono specifiche per la macchina che dovrà eseguirle. Si parla di linguaggio a basso livello. Ricordiamo ance il **linguaggio assembly** che i differenzia dal precedente in quanto le istruzioni sono espresse mediante simboli
  - **linguaggi di alto livello**: il programmatore utilizza un linguaggio vicino al problema, senza doversi preoccupare di gestire i dettagli dell'hardware del computer che dovrà eseguire le istruzioni. I programmi sono portabili, cioè possono essere utilizzabili, senza modifiche, su macchine differenti. Si parla di linguaggi alto livello.

- **linguaggi di marcatura**
  - **HTML**, dichiarativo, interpretato usato per la creazione di pagine Web, richiede la presenza di un browser per la sua interpretazione
  - **XML** consente di definire la struttura di documenti e dati

3.1.1.3. Compilatori ed interpreti

Generalmente il programmatore utilizza linguaggi ad alto livello, mentre il computer può eseguire solo istruzioni espresse in linguaggio macchina. È come far comunicare fra loro due persone che utilizzano due lingue differenti: chiaramente è necessario l'intervento di un traduttore che conosca entrambe le lingue.

Nell'ambito della programmazione abbiamo due tipi di strumenti che corrispondono a differenti modalità per la traduzione dal linguaggio ad alto livello al linguaggio macchina:

- **compilatori**: programmi che trasformano il codice sorgente (cioè il programma in linguaggio ad alto livello) prima in codice oggetto, poi in codice eseguibile scritto in linguaggio binario. Naturalmente in linguaggio macchina le istruzioni devono corrispondere ad operazioni molto semplici da seguire, quindi in generale ad una istruzione ad alto livello corrispondono più istruzioni a basso livello. Quando è terminato il processo di traduzione e viene generato l'eseguibile (file con le istruzioni in linguaggio macchina) l'utente della applicazione può chiedere al computer di eseguire le relative istruzioni.

- **Interpreti**: programmi che esaminano ed eseguono le istruzioni una per volta: la prima istruzione viene elaborata, si individuano le corrispondenti operazioni elementari in linguaggio macchina con una *traduzione al volo* e si eseguono, di seguito la seconda istruzione presa in considerazione ed eseguita, e così via. Non viene generato un codice eseguibile, il computer può eseguire le istruzioni solo in presenza del software che ha il compito di interprete.

Per alcuni linguaggi, inizialmente per Java, ma ora anche per il **C#,** ci sono due fasi distinte di traduzione:

1. **Compilazione in codice intermedio** o pseudocodice: si produce un eseguibile che però non fa riferimento a nessun hardware specifico, non può quindi essere eseguito su nessuna macchina reale, ma solo su una macchina virtuale. Tale codice intermedio viene salvato su un file ed è portabile, cioè indipendente dalla piattaforma hardware e software

2. **Interpretazione del codice intermedio** da parte di un interprete specifico per ogni differente piattaforma su cui si vuole eseguire il programma.

3.1.1.4. Grammatica dei linguaggi

Ogni linguaggio è definito da una grammatica, cioè da un insieme di regole che devono essere rispettate per scrivere programmi corretti dal punto di vista formale.

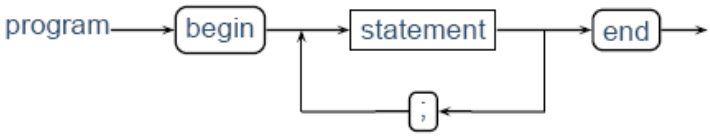
Di seguito verranno presentate le regole per scrivere in modo corretto un programma in linguaggio C# senza utilizzare, per semplicità, formalismi particolari: deve comunque essere chiaro che ogni singolo aspetto del linguaggio è definito con la massima precisione.

A solo scopo di presentazione indichiamo due dei principali formalismi utilizzati:

***Backus Naur Form* - BNF**[38]

istr-switch ::= **switch**(espr)
{ {**case** espr-cost: {istr} } [**default:** {istr}] **}**

Grafo sintattico



[38] proposto negli anni 1960 da John Backus e migliorato da Peter Naur

3.2. AMBIENTI DI SVILUPPO PER IL LINGUAGGIO C#

PREREQUISITI
Installazione sul proprio computer di Microsoft Visual C# 2008 Express Edition, oppure di SharpDeveloper: entrambi i software sono liberamente scaricabili e utilizzabili
OBIETTIVI
Saper scrivere in linguaggio C# ed eseguire semplici applicazioni Console

3.2.1. Introduzione a Microsoft® Visual C# 2008 Express Edition

Introduzione a Visual Studio, in particolare a Visual C# Express Edition. Vedi anche nel capitolo seguente l' analoga introduzione a SharpDevelop.

3.2.1.1. Ambiente di sviluppo integrato Visual C# 2008 Express Edition

Iniziamo a programmare utilizzando gli strumenti presentati (vedi capitolo 1.1.5.3 e 1.1.5.4) e supponendo quindi di avere installato sul proprio computer il software necessario, tutto gratuito e liberamene utilizzabile.

Si precisa che questo testo non intende essere un manuale di Visual C# 2008 Express Edition, semplicemente visto che *si impara facendo* è necessario riferirsi anche agli strumenti specifici che si stanno utilizzando.

Nessuna pretesa di completezza, l'idea è che le note seguenti possano essere un aiuto ed un supporto per iniziare (e continuare) a programmare.
Un suggerimento: le immagini e le osservazioni qui presentate possono servire come guida: si invita sempre a provare e a sperimentare di persona.

Avviamo quindi Microsoft Visual C# 2008 Express Edition e osserviamo come si presenta questo ambiente di sviluppo integrato (IDE): lo schema seguente guida alla osservazione della finestra che si presenta[39]

[39] Tale finestra può naturalmente essere personalizzata: in questo testo si fa sempre riferimento alle impostazioni standard, che in qualsiasi momento si possono attivare dalla barra dei menu con Finestra / Reimposta layout finestra.

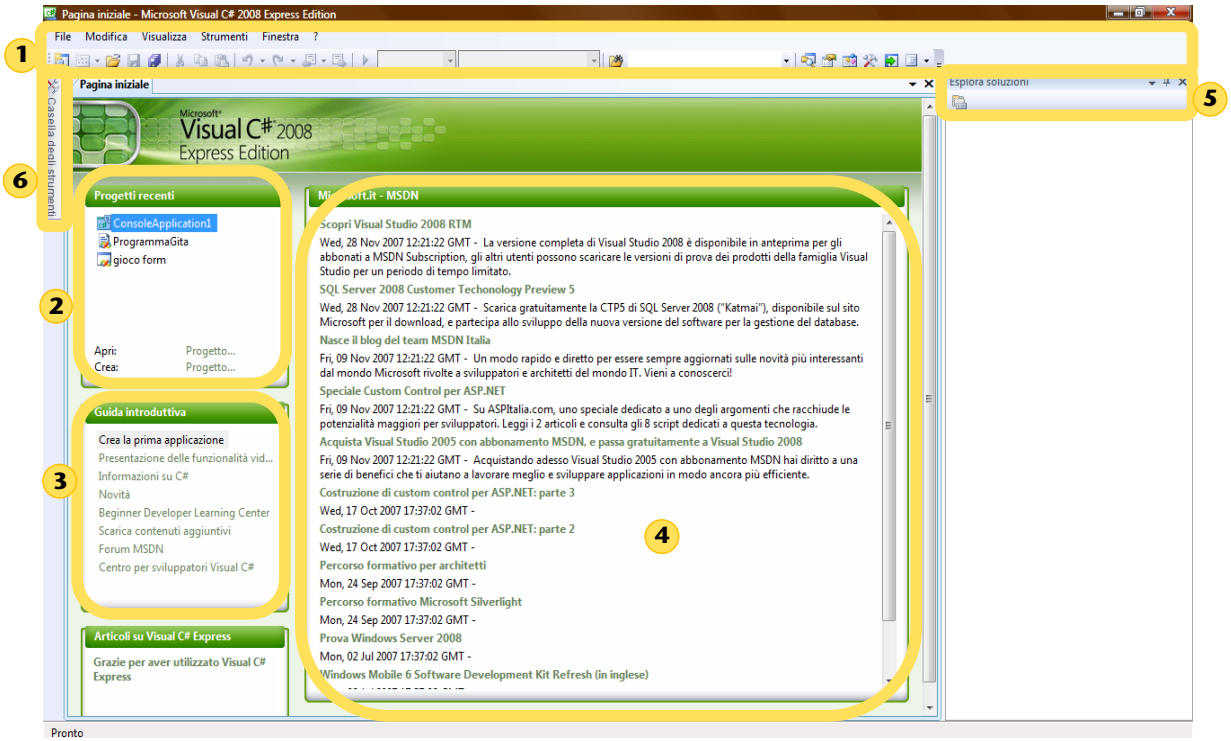
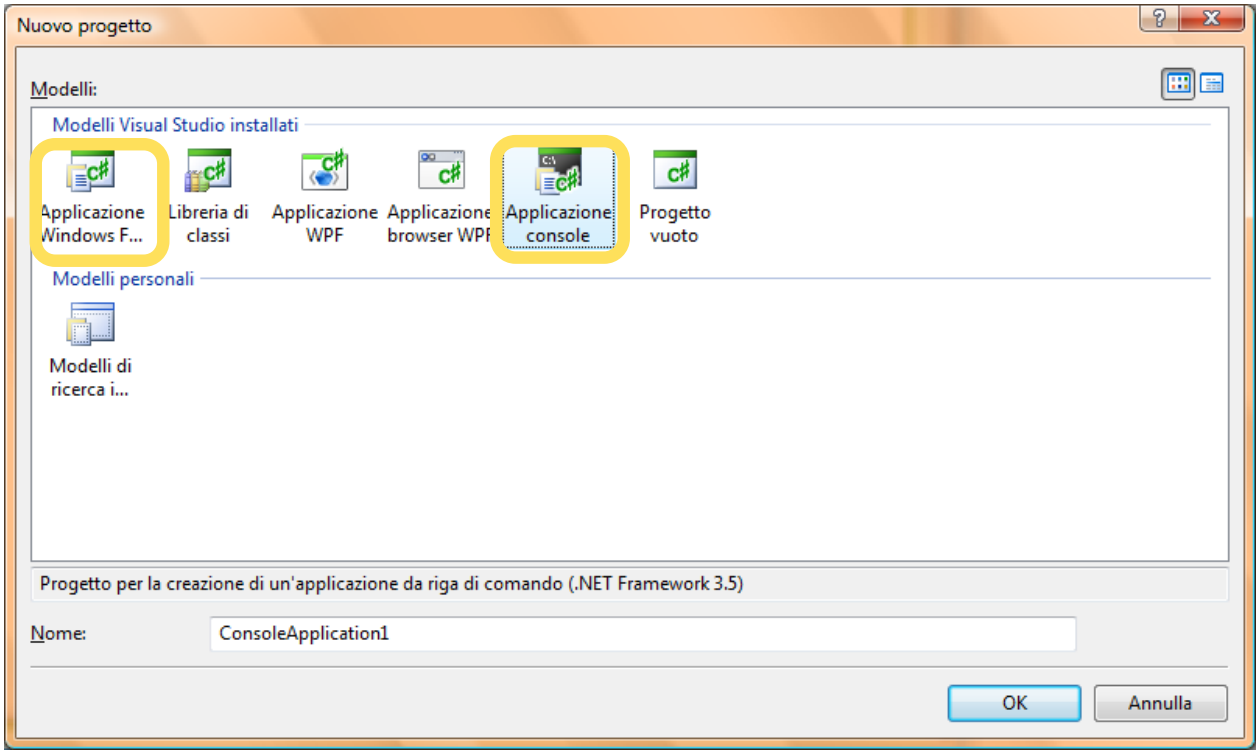


Figura 3-1 – Layout ambiente di sviluppo integrato Visual C# 2008 Express Edition

1	barra dei menu e barre degli strumenti
2	area da utilizzare per creare progetti o aprire progetti precedenti
3	area da consultare per studiare!! Contiene riferimenti importanti a strumenti di supporto sia per gli sviluppatori principianti, si a per i professionisti a qualsiasi livello (MSDN)
4	parte centrale della finestra: questo spazio contiene l'area i lavoro del programmatore, in questo momento presenta collegamenti ad informazioni ed aggiornamenti relativi al software che stiamo utilizzando
5	esplora soluzioni ci permette di lavorare direttamente sui file che fanno parte del nostro progetto (in pratica è una finestra di esplora risorse integrata nell'ambiente di sviluppo). In questo momento è vuota, perché non si sono progetti aperti, occupando spazio sullo schermo: se si preferisce si può nascondere.
6	Casella degli strumenti: contiene molti strumenti utili in particolare per la programmazione visuale (vedi 6.2): da notare che questa area ora si trova nascosta, in modo da non rubare spazio sullo schermo, ma basta passarci sopra con il mouse che si apre

3.2.1.2. Creazione progetto

Dalla area 2 della videata rappresentata nella figura precedente relativa ai progetti (oppure come sempre dalla barra dei menu) cominciamo a creare un progetto; ecco la finestra che ci si presenta:



È possibile partire da zero (progetto vuoto), oppure utilizzare modelli di Visual Studio già predisposti e installati; in figura quelli standard, fra i quali in questo volume si considerano:

- applicazione console
- applicazione Windows Form (vedi sezione 6).

Fra i modelli proposti selezioniamo per ora *applicazione console* e otteniamo la creazione del nostro **progetto**[40].

[40] Se la videata che ottieni è diversa da quella in figua, vai sul menu Finestra e scegli l' opzione *Reimposta layout finestra*, ripristinando così il layout predefinito

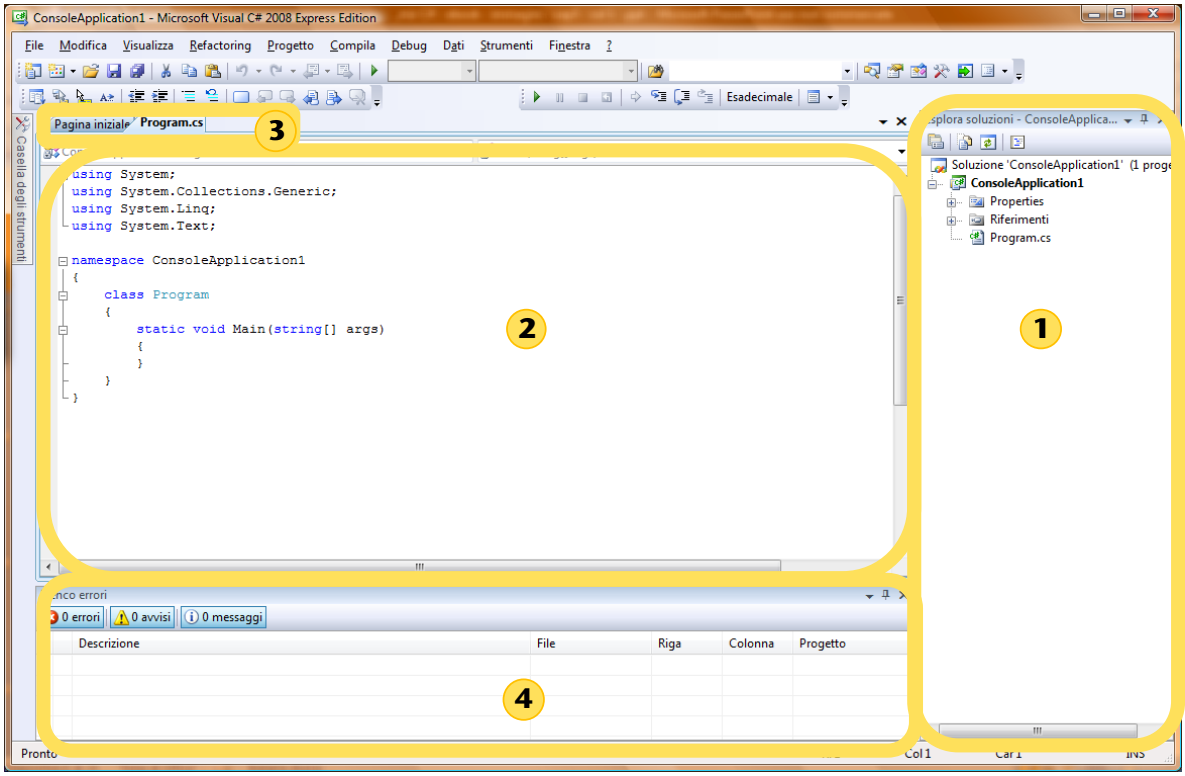


Figura 3-2 - Layout sviluppo applicazione console

1	Esplora soluzioni: il contenuto si è arricchito, troviamo tutti i file che sono stati creati per il nostro progetto. In particolare il file Program.cs contiene le istruzioni in linguaggio C# da fornire al computer.
2	nella area principale di lavoro compare ora il codice già predisposto sulla base del modello che avevamo scelto
3	qui sono indicate tutte le schede che possiamo visualizzare: in questo momento è selezionata quella di Program.cs, ed il relativo contenuto è mostrato nella area principale, come abbiamo appena visto; con click su pagina iniziale otteniamo di nuovo la videata di Figura 3-1
4	area per la segnalazione degli errori; durante l' esecuzione della applicazione si possono visualizzare i valori delle variabili locali

3.2.1.3. Sviluppo codice con IntelliSense

Molto bene! Se siamo arrivati fin qui possiamo cominciare a programmare, cioè a scrivere codice. Non è difficile: per ora forse non riusciamo a capire tutto quello che è scritto, ma non è un problema. Utilizziamo questo nuovo strumento e cerchiamo di prendere confidenza con l'IDE (ambiente di sviluppo integrato): se invece preferisci entrare prima nei dettagli della programmazione vedi paragrafo 3.3.1.1.[41]

[41] I due metodi di lavoro sono differenti ma conducono allo stesso risultato: chi prefereisce *smanettare* prosegue la lettura di questo capitolo, dopo analizza tutto in dettaglio. Chi invece è più analitico e

Intanto esaminiamo il codice che è stato preparato a seguito della nostra richiesta di utilizzare un modello per applicazioni Console:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         4
      }
   }
}
```

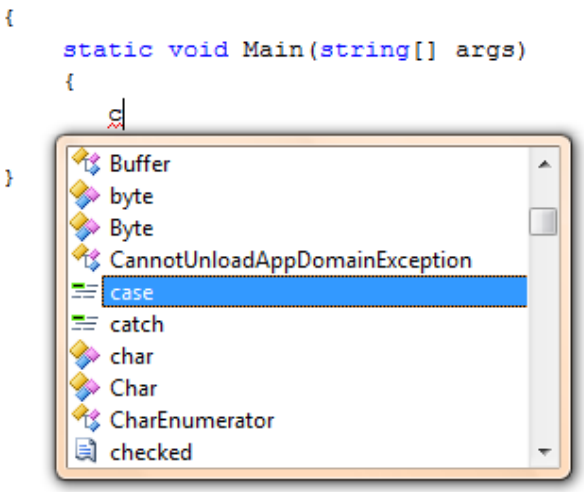
Sposta il cursore nella posizione 4 , premi Invio e prova a scrivere il seguente testo:

```
Console.WriteLine("Ciao");
```

Tutto bene? Proviamo a riflettere insieme su ciò che può essere accaduto: appena scriviamo la lettera C l'ambiente integrato interviene a guidarci nel nostro lavoro.
Possiamo fare tutto da soli e scrivere la parola completa, ma possiamo scegliere di avvalerci dei suggerimenti (scorri in basso con le frecce, troverai proprio la parola Console, premi Invio e viene automaticamente inserita). Lo stesso aiuto ci viene proposto per ogni elemento del linguaggio.

L'aiuto che ci viene offerto copre due aspetti importanti:

- completamento automatico (con il cursore scendiamo fino a trovare la parola che ci interessa, quindi basta premere invio invece di digitare tutto il testo
- visualizzazione del riferimento al manuale del linguaggio relativo alla nostra specifica attività: si impara facendo!



Il termine **IntelliSense** indica questo insieme di funzionalità.

Di seguito proviamo a scrivere la seconda istruzione: `Console.ReadLine();`

---

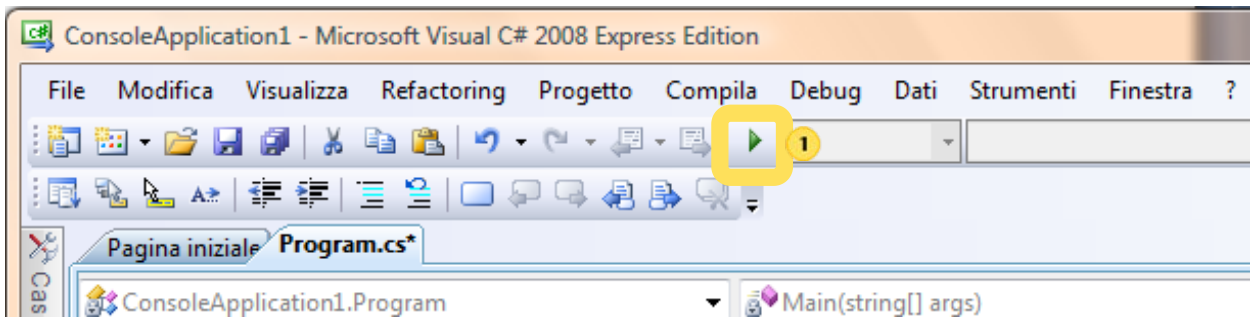
preferisce tenere subito sotto controllo tutti i gli aspetti della codifica va prima a ... studiare, poi torna indietro per provare ad applicare. È invece importante che ciascuno riesca a seguire le proprie curiosità e ad individuare qual è per sé stesso il metodo di lavoro più interessante e quindi efficace. È questo uno dei segreti per diventare buoni programmatori.

3.2.1.4. Debug ed esecuzione

Si tratta di provare a far eseguire al computer le istruzioni che abbiamo scritto: come si è visto ci sono due distinte fasi, la prima (compilazione) consiste nella traduzione delle nostre istruzioni dal linguaggio C# ad un linguaggio più vicino alla macchina con la generazione di un programma in linguaggio MSIL (Microsoft Intermediate Language), la seconda consiste nell'interpretazione di questo codice intermedio, cioè nell'esame una per una di ogni singola istruzione con relativa traduzione in linguaggio macchina ed esecuzione.

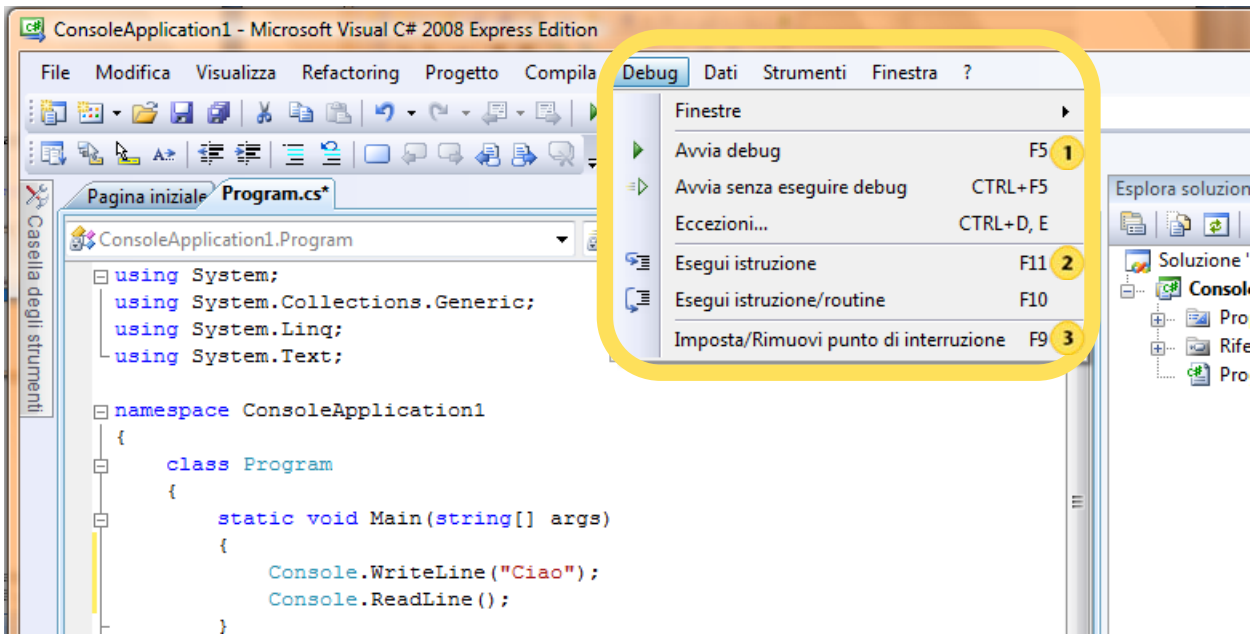
Una osservazione: verifichiamo di aver scritto correttamente le istruzioni in C#, altrimenti la prima fase, invece di tradurre e generare il codice intermedio, si limita a produrre la lista degli errori riscontrati, ahimè!

Operativamente si tratta di avviare il **debug**, cioè la ricerca degli eventuali errori (bug) e, se tutto va bene, la compilazione. Ci sono diverse possibilità:

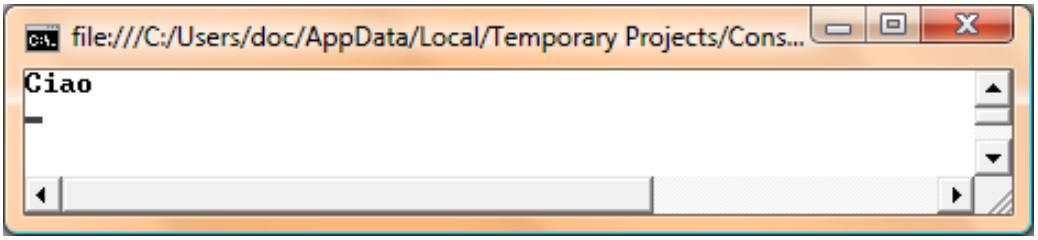


Come mostrato nella figura qui sopra, dalla barra degli strumenti standard , click sul pulsante con la freccia verde Avvia Debug (oppure F5).

Oppure, come nella finestra qui sotto barra dei menu/debug e posso scegliere ancora Avvia debug (f5) ma anche Esegui istruzione (f11) per eseguire, e quindi controllare, una istruzione per volta, o ancora eseguire tutto fino ad uno specifico punto i interruzione fissato dl programmatore che può così controllare specificatamente l' esecuzione di particolari istruzioni di suo interesse.



Per ora la procedura più semplice (1) e dovremmo ottenere una finestra come la seguente:



Nelle applicazioni Console come la nostra l'interazione fra il computer e l'utente durante l'esecuzione del programma avviene attraverso una interfaccia costituita da una finestra a riga di comando, in cui ci si alterna a scrivere riga per riga : il computer richiede eventuali dati che devono essere inseriti dall'utente con dei messaggi, l'utente risponde inserendo i dati in genere da tastiera (i valori inseriti vengono comunque visualizzati) , il computer infine mostra i risultati attesi.

In questo primo esempio molto semplice il computer invia il messaggio a video per l'utente: dopo aver letto possiamo chiudere la finestra.

In caso contrario ... s i correggono gli errori.

3.2.1.5. Correzione degli errori

Ci sono diversi tipi di segnalazioni:

- **messaggi**
- **avvisi** (Warning) cioè segnalazioni di situazioni anomale;
- **errori** di sintassi: il codice non è corretto, non rispetta le regole del linguaggio, non si può procedere alla compilazione. Osserva che comunque quando il codice presenta errori, già mentre si scrive, viene segnalata in rosso la necessità di intervenire con una correzione.

L'IDE di Visual Studio ci aiuta per la correzione di questi errori: prepariamo l'ambiente di lavoro (vai sulla barra dei menu/Visualizza/ Elenco errori) e facciamo subito una prova. Per esempio cancella il punto e virgola alla fine della prima istruzione che hai scritto e osserva i messaggi che vengono visualizzati: doppio click sulla riga della finestra errori che contiene la segnalazione e si viene subito portati sulla riga di codice da correggere!

Possono però essere presenti altri tipi di errori:

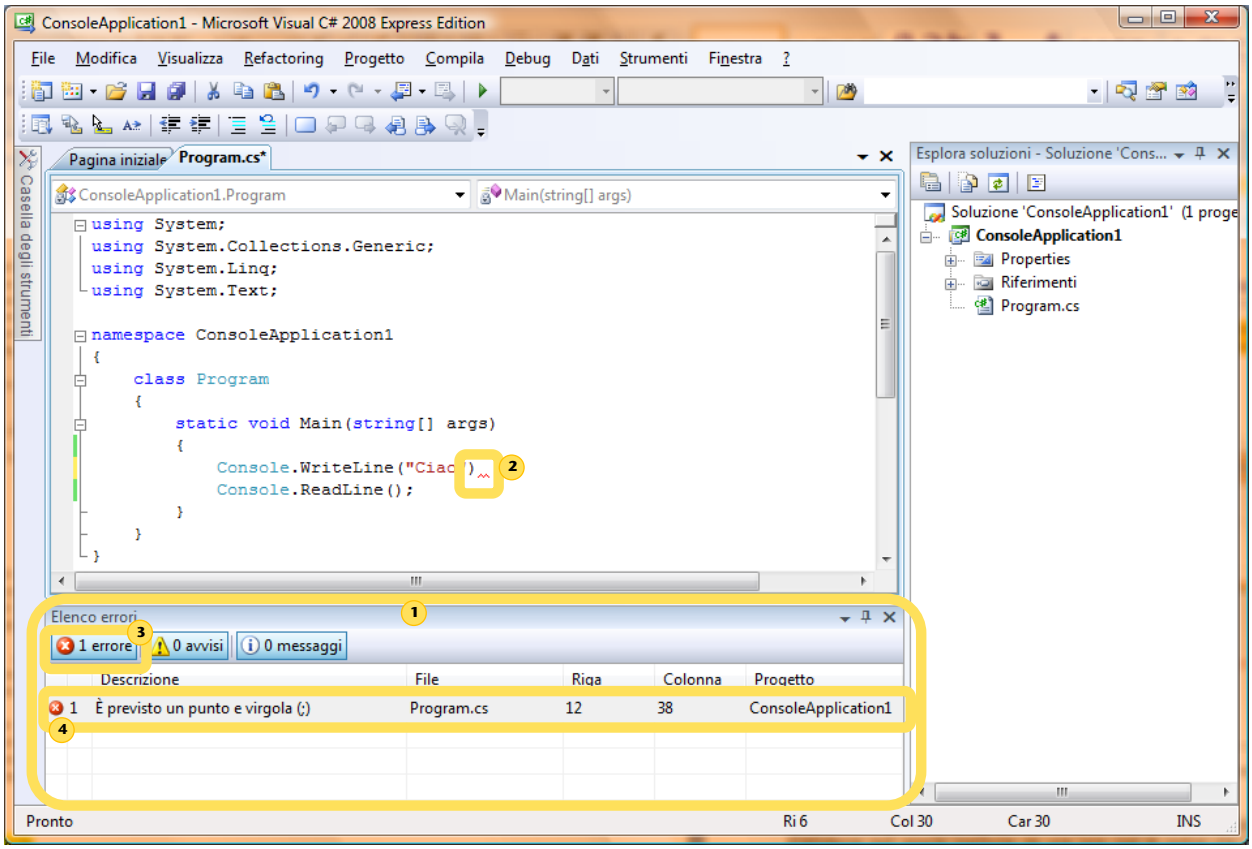
- **errori di logica**: il programma viene compilato ed eseguito, ma i risultati non sono corretti. In questo caso c'è probabilmente un errore già nell'algoritmo, che se necessario bisogna correggerlo, quindi correggere anche il programma, quindi provare a far di nuovo eseguire.

- **Errori a tempo di esecuzione**: viene interrotta l' esecuzione del programma.

Non vorremmo qui dover parlare degli **errori di analisi**: il codice viene compilato ed eseguito, i risultati sono coerenti, ma ci si accorge solo ora che per un fraintendimento, per una lettura affrettata del testo, si è risolto un problema che non corrisponde, almeno in parte , a quello posto (ad esempio si chiede di calcolare perimetro ed area del rettangolo, si calcolano invece perimetro e area del quadrato).

È chiaro che tanto più tardi si scopre un errore, tanto più gravi sono le conseguenze, perché si dovranno ripetere tutte le fasi di lavoro successive all'errore stesso. D'altra parte un buon metodo di lavoro ci dovrebbe portare ad avere, in questa fase, solo eventuali errori di sintassi.
Per la correzione di tali errori l' IDE di Visual Studio ci aiuta: facciamo subito una prova, ma prima di tutto prepariamo l' ambiente di lavoro: vai sulla barra dei menu/Visualizza/ Elenco errori , quindi per esempio cancelliamo il punto e virgola alla fine della prima istruzione che abbiamo scritto.

Osserviamo insieme:



1	finestra Elenco errori, che segnale errori veri e propri (devono essere corretti per consentire la compilazione), avvisi cioè avvertimenti (warning), segnalazioni di situazioni anomale, ed infine semplici messaggi
2	quando il codice presenta errori, già mentre si scrive, viene segnalata in rosso la necessità di intervenire con una correzione
3	quando si scrive del codice non corretto viene indicato qui il numero di errori riscontrati: in questo caso 1 errore, nessun avviso e nessun messaggio
4	per ognuno degli errori riscontrati una riga di dettaglio e di spiegazioni: doppio click su questa riga e ci ritroviamo automaticamente sulla riga di codice da correggere

Correzione degli errori: Si Tratta di una attività molto importante, da non sottovalutare. È bene individuare soluzioni corrette da un punti di vista logico a riuscire a scrivere buon codice, ma ciò non toglie che un buon programmatore sia tale solo se sviluppa anche una buona strategia per individuare e correggere gli errori.

3.2.2. Introduzione a SharpDevelop 3.0

Introduzione a SharpDevelop : vedi anche l' analoga introduzione a nel Visual C# Express Edition capitolo precedente dove sono stati introdotti alcuni concetti che qui non venono ripetuti, semplicemente se ne mostra l' implementazione nel nuovo Ambiente di Sviluppo Integrato (IDE).

3.2.2.1. Ambiente di sviluppo integrato SharpDevelop 3.0

Presentiamo qui un altro ambiente di sviluppo adatto alla programmazione in linguaggio C# : si suggerisce di leggere questa parte dopo aver installato il software necessario, rilasciato sotto i termini della licenza GNU (vedi paragrafo 1.1.5.7) e di cui si raccomanda il download (vedi paragrafo 1.1.5.6). Anche in questo caso non si intende presentare un manuale di SharpDevelop, bensì introdurre all' uso di uno strumento che può essere assai utile per imparare a programmare.

Avviamo SharpDevelop e osserviamo la struttura della videata iniziale:

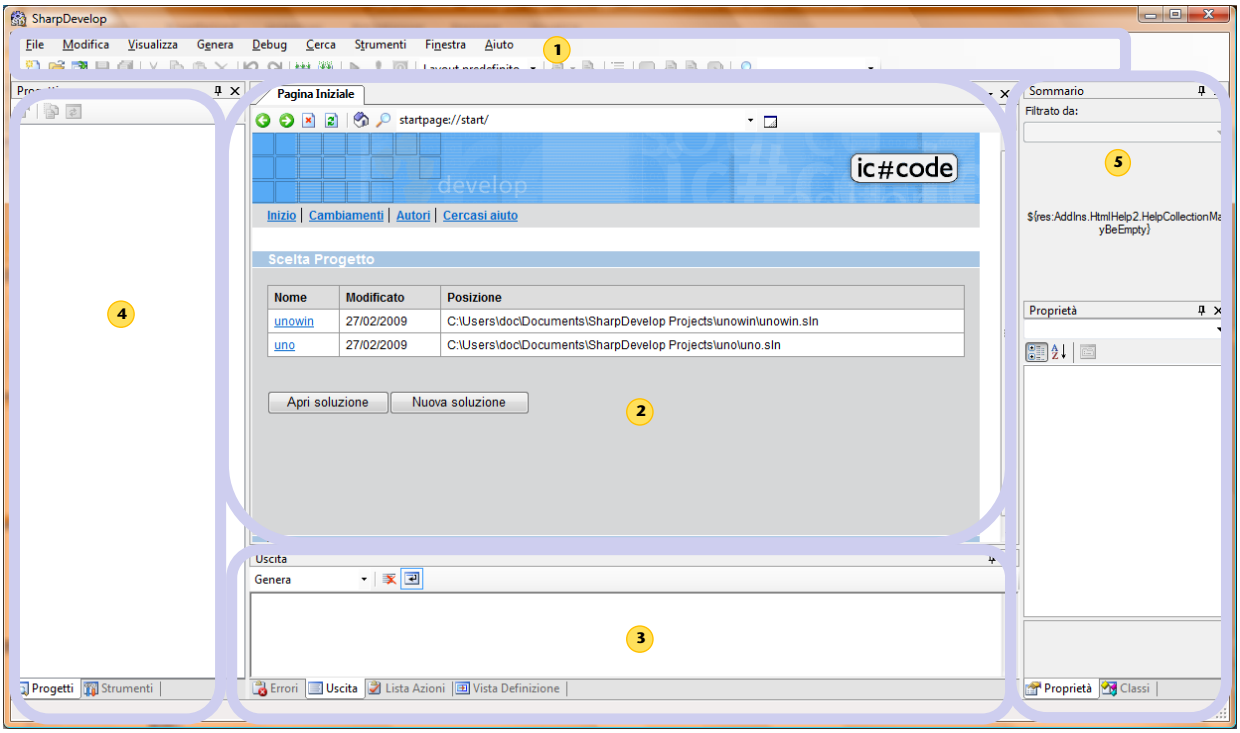


Figura 3-3 - Layout ambiente di sviluppo integrato SharpDevelop 3.0

1	barra dei menu e barre degli strumenti
2	area da utilizzare per creare progetti o aprire progetti precedenti
3	area per la segnalazione degli errori; durante l' esecuzione della applicazione si possono visualizzare i valori delle variabili locali
4	struttura del progetto
5	sommario

Creiamo il nostro primo progetto: dall' area centrale il pulsante *Nuova soluzione* apre la finestra *Nuovo progetto* in cui possiamo indicare le specifiche del progetto che andiamo a realizzare: scegliamo la *categoria C#*, quindi *applicazioni Windows* (in seguito si vedrà ASP.NET per le applicazioni WEB); ora fra i *modelli* scegliamo *Applicazione Console*. L'IDE ha generato lo scheletro della applicazione Console:

```
/*
 * Creato da SharpDevelop.
 * Utente: doc
 * Data: 27/02/2009
 * Ora: 16.54
 *
 * To change this template use Tools | Options | Coding | Edit Standard Headers.
 */
using System;
namespace uno
{
	class Program
	{
		public static void Main(string[] args)
		{
			Console.WriteLine("Hello World!");

			// TODO: Implement Functionality Here
			1
			Console.Write("Press any key to continue . . . ");
			Console.ReadKey(true);
		}
	}
}
```

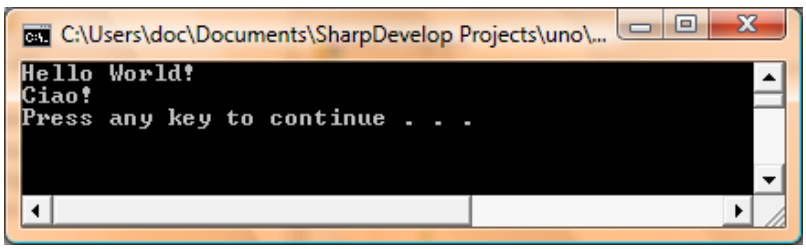
Proviamo a personalizzare un po' questo programma: sistema il cursore nella posizione 1 e scrivi la seguente istruzione

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

Osserva che anche in questo caso abbiamo il completamento automatico del codice (vedi analoga funzionalità in ambiente Microsoft descritta al paragrafo 3.2.1.3).
Non ci resta che provare ad eseguire l' applicazione:

- barra degli strumenti pulsante con la freccia verde (Esecuzione)
- tasto funzione F5
- barra dei menu / Debug
  - Esegui
  - Attiva/disattiva BreakPoint

Di seguito la videata che si ottiene ... se tutto ha funzionato!



In caso contrario si procede alla correzione degli errori.

Come già mostrato con Visual C# anche qui vengono segnalate anomalie di tipo differente, in particolare: errori, segnalazioni (warnings) e messaggi.

3.3. INIZIAMO A PROGRAMMARE

PREREQUISITI

Conoscere gli algoritmi strutturati, saper utilizzare l'editor ed il debug di Visual C# 2008 Express

OBIETTIVI

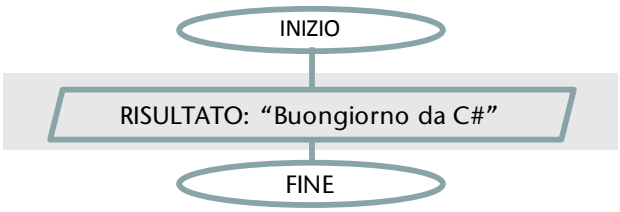
Saper scrivere in linguaggio C# ed eseguire applicazioni Console

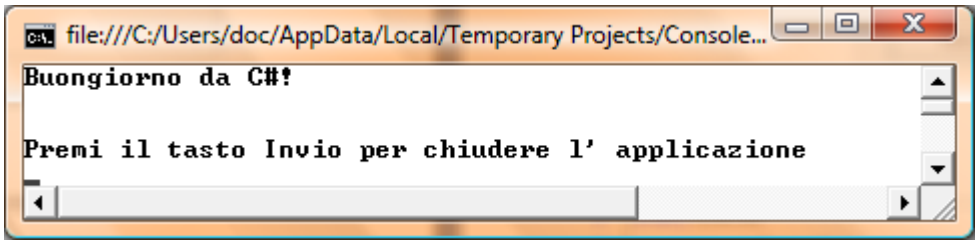
3.3.1. I primi programmi in linguaggio C#

Il primo programma "Buongiorno da C#, poi una introduzione alla gestione dell'Input/Output.

3.3.1.1. Buongiorno da C#

Osserva e leggi il codice corrispondente






```
1  using System;
2  namespace informatica
3  {
4      class esempio_01
5      {
6          static void Main()
7          {
8              Console.WriteLine ( "Buongiorno da C#!");
9              Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
10             Console.ReadLine();
11         }
12     }
13 }
```

Figura 3-4 Buongiorno da C#

Il problema che abbiamo risolto è molto semplice: chiedere al computer, con un programma scritto in C#, di visualizzare un saluto. Questo inizio è un classico di ogni lavoro sui linguaggi di programmazione, un modo per ricordare Brian Kernighan e Dennis Ritchie che per primi presentarono questo esempio nel celebre libro " *The C Programming Language* " (noto come il C di K&R).

L'algoritmo corrispondente, altrettanto semplice, è indicato in figura, seguito dalla finestra che si ottiene dall'esecuzione del relativo codice.

Proviamo ora ad esaminare tale codice e a prendere confidenza con la struttura di un programma scritto in linguaggio C#.

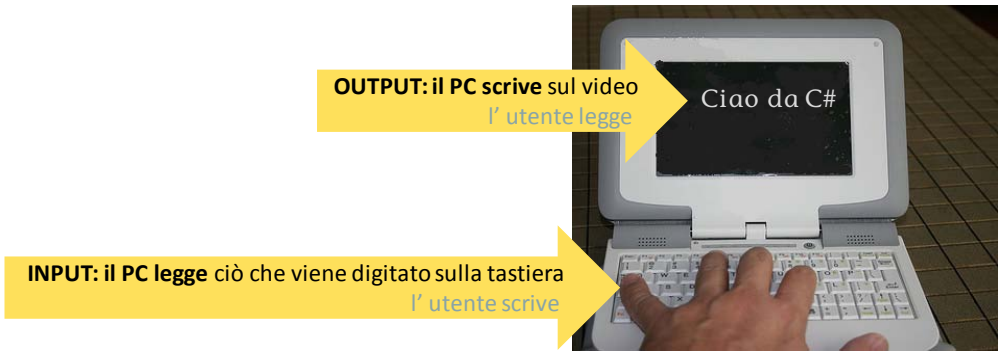
Nella rappresentazione grafica dell'algoritmo è stato evidenziato con uno sfondo più scuro il blocco che contiene le istruzioni da eseguire (una sola a dire il vero): lo sfondo più scuro individua nel codice le istruzioni corrispondenti.

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 7	contiene la parentesi graffa aperta che corrisponde al blocco di inizio dell' algoritmo; nella riga 11 troviamo la corrispondente parentesi graffa chiusa analoga al blocco fine. **Una coppia di parentesi graffe individua in C# un blocco di codice**: andiamo ad osservare l' intestazione di tale blocco, che si trova nella riga precedente, come se fosse un titolo;
riga 6	Main indica che siamo nella parte principale: precisamente **Main** indica il **punto di ingresso del programma**, cioè il punto che contiene la prima istruzione che viene eseguita all' avvio della applicazione. Il blocco del Main è costituito dalle istruzioni comprese fra le parentesi graffe aperte e chiuse che seguono questa riga ; prima di Main dobbiamo scrivere static void: per ora un po' di pazienza, impariamo a scriverlo anche senza riuscire ancora a comprenderne il significato
riga 8	Ecco la prima **istruzione** che viene eseguita: essa produce la visualizzazione sul video del computer della scritta **Buongiorno da C#**. Osserviamola in dettaglio**:** • **Console**: rappresenta il nostro Computer • **WriteLine( )**: è il messaggio che inviamo alla Console, chiediamo di scrivere una linea (Write) e poi di spostarsi sulla linea successiva; • "Buongiorno da C#" : indichiamo fra doppi apici il testo che vogliamo venga scritto sul video • Osserva: l' istruzione viene sempre chiusa da un segno di **punto e virgola**
riga 9	viene scritto a video il testo *Premi il tasto invio per chiudere l' applicazione* ma, come puoi osservare dalla figura, sono state lasciate due righe bianche al di sopra: si è ottenuto questo con **\n** ripetuto due volte prima del testo da visualizzare
riga 10	**Console.ReadLine()** in questa istruzione viene inviato un nuovo messaggio alla Console, cioè al nostro computer: viene chiesto di leggere una riga. Cerchiamo di capire: dove può essere scritta questa riga? Chi la scrive? Cosa contiene? Come si fa a capire quando la linea è terminata? La riga in questione viene scritta da colui che sta eseguendo l' applicazione utilizzando la tastiera: si può scrivere qualsiasi cosa (anche niente) e si deve concludere con il tasto Invio: è questo che indica la fine della linea, ed a quel punto il computer legge il testo che è stato scritto. Questo programma è molto semplice, per cui qualunque cosa scriviamo essa non viene ulteriormente elaborata. **Nota bene**: questa istruzione è molto utile perché obbliga il computer ad attendere l' input da parte dell' utente prima di andare avanti: se non ci fosse Console.ReadLine() il

programma dopo l' istruzione della riga 9 andrebbe subito avanti e, non trovando in questo caso altre istruzioni, terminerebbe senza dare all' utente il tempo di osservare quanto visualizzato video. In questo modo invece l' utente legge il video con calma e, quando vuole, preme invio: solo a quel punto termina il programma.

riga 11 presenta la parentesi graffa che chiude il blocco di istruzioni del Main: la nostra applicazione viene terminata.



Osserviamo ancora:

righe 6-11 in queste righe abbiamo la **definizione del metodo Main**, costituita dall' **intestazione** (riga 6) e dal **corpo** (righe 7-11) che contiene le istruzioni;

righe 4-12 in queste righe abbiamo la **definizione della classe** di nome esempio _01, costituita dall' intestazione (riga 4) e dal corpo (righe 5-12). La classe è un *contenitore* all'interno del quale in questo caso, per semplicità, è presente il solo metodo Main

righe 2-13 in queste righe abbiamo la definizione del nostro **spazio dei nomi** che abbiamo chiamato Informatica, costituito dall' intestazione (riga 2) e dal corpo (righe 3-13). Lo spazio dei nomi è un ambiente all'interno del quale in questo caso, per semplicità, è presente la sola classe esempio_01.

riga 1 **using System**: specifichiamo che intendiamo utilizzare non solo il nostro spazio dei nomi, quello che contiene cioè tutti i nomi che definiamo noi (ad esempio il nome della nostra classe che si chiama esempio_01), ma anche altri nomi che corrispondono a classi e metodi già definiti nei relativi namespace ed a cui noi vogliamo fare riferimento nel nostro programma per poterli utilizzare. Senza la clausola using System per utilizzare questi componenti dovremmo sempre specificare il nome per esteso, ad esempio System.Console.ReadLine( ), oppure System.Console.Writeline( )

riga 2 **namespace** informatica: definiamo un nostro spazio dei nomi, cioè il contesto nel quale andiamo ad operare.

Fai attenzione: il linguaggio C# è **case-sensitive**, cioè devi fare attenzione alle lettere maiuscole e minuscole, ad esempio Main e main sono differenti! Viene utilizzato uno stile *camel-case*: in ogni parola la lettera iniziale è maiuscola es. **C**onsole.**W**rite**L**ine .

Classi e **metodi** sono strutture fondamentali della **programmazione ad oggetti**: ora impariamo ad utilizzarli, anche senza comprenderne a fondo il significato, di seguito si vedranno tutti gli aspetti teorici. Tale comportamento è analogo ad esempio a quello di chi impara a guidare un'automobile: probabilmente non conosce tutti i dettagli della meccanica dell' auto, eppure imparando a guidarla farà esperienza, di seguito se è interessato potrà aprire il cofano e vedere cosa c'è dentro. Si tratta di due competenze differenti: ora a noi interessa imparare a programmare utilizzando componenti che sono già a nostra disposizione, senza essere distratti da troppi dettagli; di seguito andremo a vedere bene tutti gli aspetti.

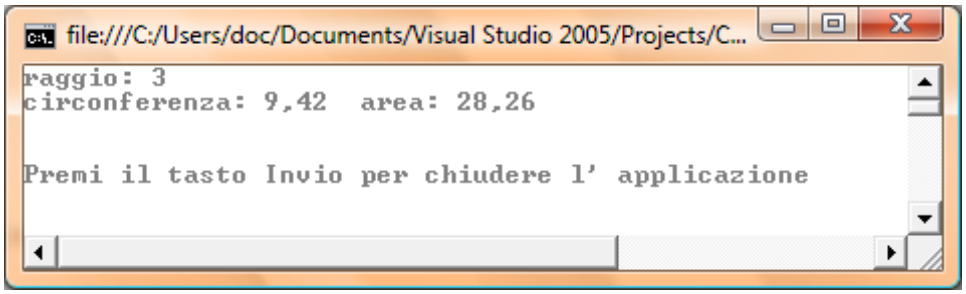
Ricorda i seguenti termini

using	direttiva per utilizzare i tipi in uno spazio di nomi , senza dover qualificare lo spazio dei nomi stesso
System	spazio dei nomi in cui sono contenute le classi che vengono comunemente utilizzate **using.System** ci consente di utilizzare Console senza dover specificare ogni volta System
namespace	parola chiave viene utilizzata per dichiarare uno **spazio dei nomi** , cioè l' ambito in cui si opera (serve a raggruppare tutti i nomi che utilizziamo ed a evitare eventuali omonimie)[42] quando il programmatore non dichiara in modo esplicito uno spazio dei nomi il compilatore aggiungerà uno spazio dei nomi predefinito: ciò garantisce
class	descrizione di un oggetto. L' oggetto che descriviamo (in questi casi semplici) è la nostra applicazione, quindi la classe sarà un contenitore di tutto ciò che la costituisce. Deve sempre essere presente il metodo Main.
Main( )	è il metodo, cioè il modo, per far eseguire il programma. Rappresenta il punto di ingresso dell' esecuzione della nostra applicazione, cioè contiene la prima istruzione che viene eseguita
static void	parole chiave che devono precedere l' identificatore Main: in questo momento un po' di pazienza, in seguito verrà spiegato il significato di questi termini
Console	rappresenta il nostro computer
WriteLine( )	metodo, cioè il modo, per scrivere un testo: ad esempio Console.WriteLine() consente alla Console, a cui viene inviato il messaggio, di scrivere il testo. Il testo compare sul video (e l' utente della applicazione lo può leggere).
ReadLine( )	metodo per leggere un testo, ad esempio Console.ReadLine( ) consente alla Console, a cui viene inviato il messaggio, di leggere il testo. Il testo viene in questo caso digitato da tastiera (dove viene scritto dall' utente della applicazione)
( )	una coppia di parentesi tonde indica che l' identificatore che la precede è un metodo Esempio: Main( ), WriteLine( ), ReadLine( )
{ }	una coppia di parentesi graffe delimita un blocco (un insieme) di istruzioni
istruzione (statement)	comando che viene impartito al computer. U una istruzione viene terminata dal punto e virgola

[42] Questo è un concetto importante in informatica, per comprendere bene osserviamo il mondo reale: due persone si chiamano Giuseppe, come distinguerle? Facciamo riferimento al contesto: Giuseppe, il mio compagno di di classe, oppure Giuseppe, che gioca con me a pallavolo. Se specifico il contesto, ad esempio Pallavolo, potrò parlare di Giuseppe senza ambiguità. La parola contesto è generica, in termini tenici si parla di *spazio dei nomi*, lo spazio (pallavolo) all' interno del quale quel nome (Giuseppe) è ben identificato perché univoco: dovrei dire *using pallavolo*!

3.3.1.2. Variabili e costanti

Osserva e leggi il codice corrispondente




```
1   using System;
2   namespace informatica
3   {
4       class esempio_02
5       {
6           static void Main()
7           {
8               const double piGreco = 3.14;
9               int raggio = 3;
10              double circ, area;
11              circ = raggio * piGreco;
12              area = piGreco * raggio * raggio;
13              Console.WriteLine ( "raggio: {0} ",raggio);
14              Console.WriteLine("circonferenza: {0}  area: {1}", circ, area);
15              Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
16              Console.ReadLine();
17          }
18      }
19  }
```

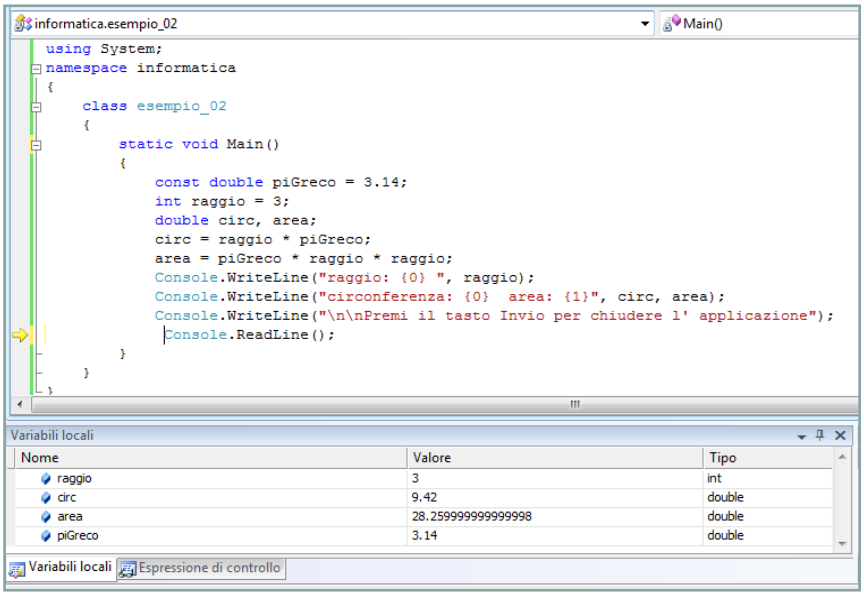
Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 8 in questo programma cominciamo ad utilizzare variabili e **costanti** e questa riga contiene la prima istruzione che viene eseguita, osserviamola in dettaglio**:**

- **const**: parola chiave utilizzata per specificare che il valore della variabile a cui si riferisce è costante e quindi non può essere modificato
- **double**: indica il tipo di dato che intendiamo utilizzare; double significa che ci interessa gestire valori numerici con parte intera e parte decimale
- **piGreco** : identificatore della nostra variabile, cioè il nome che attribuiamo a tale variabile e che ci servirà per riferirci ad essa nelle diverse istruzioni del nostro programma. Fai attenzione: prima è stato usato lo specificatore const, quindi il valore sarà costante, cioè non potrà essere modificato nel corso del programma
- **= 3.14 :** inizializzo piGreco con il valore 3.14

riga 9	dichiarazione, definizione ed inizializzazione di una **variabile** di nome raggio, osserviamo in dettaglio**:** • **int**: indica il tipo di dato che intendiamo utilizzare; int significa che ci interessa gestire valori numerici di tipo intero (senza parte decimale) • **raggio**: : identificatore della nostra variabile • **= 3:** inizializzo raggio con il valore 3
riga 10	dichiarazione e definizione di due variabili di tipo double (valori numerici con parte decimale), di nome rispettivamente circ e area che non vengono inizializzate, cioè non si specificano dei valori iniziali**;**
riga 11	viene calcolata la misura della circonferenza e vi è l' **assegnazione** del risultato alla variabile circ**;** nella riga seguente si calcola il valore del' area;
riga 13-14	Console.WriteLine(. . .) → abbiamo l' output a video, osserviamo in dettaglio**:** • ("raggio: ") il testo compreso fra apici viene visualizzato in output, quindi in questo modo possiamo mandar dei messaggi predisposti dal programmatore, ma non potremo inviare in output il risultato di un calcolo. È necessario specificare che vogliamo visualizzare , in questo caso, il valor della variabile raggio • ("raggio: { } ")nella riga 13 abbiamo inserito le parentesi graffe che rappresentano un segnaposto, indicano cioè che in quella posizione andrà inserito un valore (il valore di una variabile, o eventualmente il risultato di una espressione). • ("circonferenza: { } area: { }") nella riga 14 abbiamo inserito due coppie di parentesi graffe, che rappresentano i segnaposti per indicare dove andranno inseriti i valori delle corrispondenti variabili. Tutto OK, ma come facciamo ad indicare con precisione quale valore visualizzare, evitando di confondere la circonferenza con l' area? È chiaro quello che noi vogliamo ottenere: nella prima posizione ci deve essere il valore della circonferenza, nella seconda quello dell' area • ("circonferenza: {0} area: {1}", circ, area) questa è l' indicazione completa. Dopo i doppi apici indichiamo la lista delle variabili di cui vogliamo visualizzare i valori, all' interno di ogni coppia di parentesi graffe indichiamo quale variabile della lista vogliamo considerare specificandone la posizone nella lista stessa. Ricordiamoci: il computer inizia a contare da zero, quindi le posizioni nel nostro esempio sono nell' ordine 0 (in corispondenza della variabile circ) e 1 (per la variabile area).

Prova ad eseguire passo passo le istruzioni del listato proposto (ad esempio con Visual C# Express Edition esegui con F11) ed osserva i valori delle variabili locali.

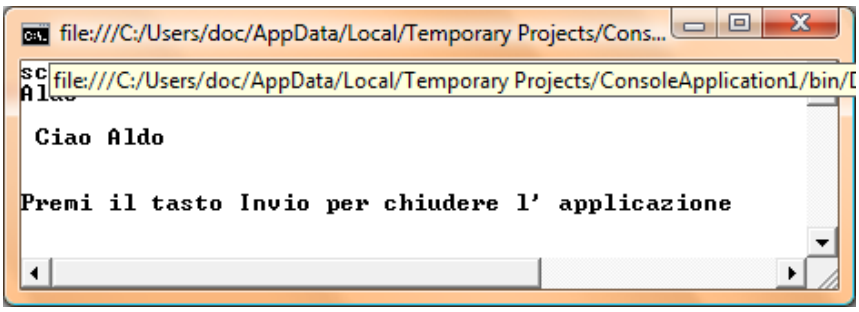


Ricorda i seguenti termini o espressioni[43]

Int, double	Specifica il tipo di dato, cioè il modo in cui verranno memorizzati i dati delle variabili di quel tipo e le operazioni che si possono effettuare su di essi
const	parola chiave utilizzata per specificare che il valore della variabile a cui si riferisce è costante e quindi non può essere modificato
Locazione di memoria	Area della memoria RAM in cui sono memorizzati i dati durante l'esecuzione del programma
variabile	Area di memoria RAM identificata da un nome ed in cui viene memorizzato un valore che può essere modificato durante l'esecuzione del programma. Identificatore di una variabile: nome della variabile Tipo di una variabile: tipo del valore associato alla variabile, ad esempio valore numerico di tipo intero, valore numerico con parte decimale, carattere, stringa(sequenza di caratteri) e così via. I valori vengono memorizzati con tecniche differenti, specifiche per ogni tipo di dato[44]. Valore di una variabile: valore memorizzato nella locazione della RAM, ad esempio 5, oppure 7.2, oppure "Ciao" Indirizzo di una variabile: indirizzo di memoria della locazione RAM in cui viene memorizzato il valore, esempio 0x76315417654 Dichiarazione di una variabile: introduco la variabile nel mio programma e la descrivo specificandone nome e tipo, ma senza allocare spazio in memoria Definizione di una variabile: creazione della variabile, cioè allocazione dello spazio in memoria Inizializzazione di una variabile: memorizzazione di un valore specifico contestualmente alla creazione della variabile stessa
costante	Area di memoria RAM identificata da un nome ed in cui viene memorizzato un valore che non può essere modificato durante l'esecuzione del programma.

3.3.1.3. Input da tastiera

Osserva e leggi il codice corrispondente



[43] Nella prima colonna sono indicate con sfondo grigio le parole chiave del linguaggio C#, con sfondo colorato le parole che corrispondono a concetti importanti relativi all' argomento che stai studiando.

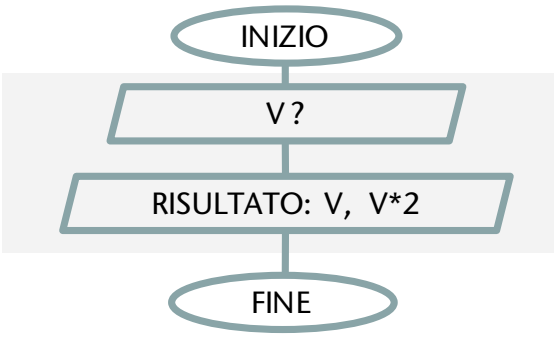
[44] Questo aspetto verrà discusso in seguito in modo approfondito, ora è sufficiente intuire il concetto.

```
1  using System;
2  class esempio
3  {
4     static void Main()
5     {
6        string nominativo;
7        Console.WriteLine("scrivi il tuo nome");
8        nominativo = Console.ReadLine();
9        Console.WriteLine("\n Ciao {0} ", nominativo);
10       Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
11        Console.ReadLine();
12    }
13 }
```

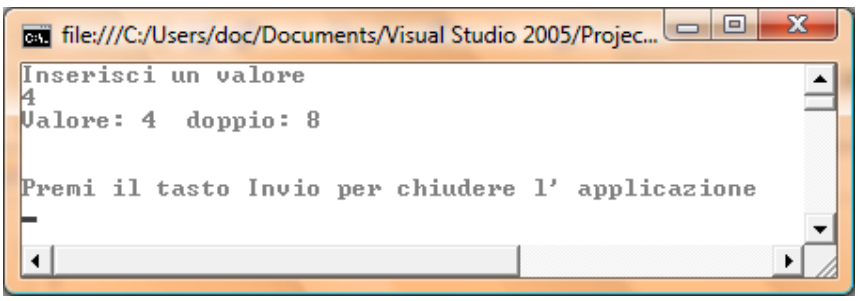
Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 6	dichiarazione e definizione di una variabile di nome nominativo e di tipo stringa, adatta cioè a memorizzare una o più parole**;**
riga 8	il valore digitato a tastiera dall' utente viene acquisito e memorizzato nella variabile nominativo
riga 9	viene mandato a video un saluto personalizzato.

Problema→ Dato un valore specificato dall'utente calcolare il doppio



Osserva e leggi il codice corrispondente



```
1  using System;
2      namespace informatica
3      {
4          class esempio_03
5          {
6              static void Main()
7              {
8                  int V;
9                  string temp;
10                 Console.WriteLine ( "Inserisci un valore ");
11                 temp = Console.ReadLine();
12                 V = Convert.ToInt32(temp);
13                 Console.WriteLine("Valore: {0}  doppio: {1}", V, V*2);
14                 Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
15                 Console.ReadLine();
16             }
17         }
18     }
```

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

Righe 8-9 vengono dichiarate due variabili: V di tipo intero , come ci si poteva aspettare dall' algoritmo, ma anche una variabile di tipo stringa, destinata cioè a contenere un testo

riga 11: il valore inserito da tastiera da parte dell' utente, cioè un numero, viene acquisito e memorizzato su una variabile di tipo stringa: questo perché un input da tastiera è sempre interpretato come sequenza di caratteri e codificato come tale (vedi paragrafo 4.1.2.1)

riga 12: `V = Convert.ToInt32(temp)` il valore inserito da tastiera da parte dell' utente e memorizzato sulla variabile temp di tipo stringa viene convertito in un intero; , osserviamo in dettaglio**:**

- `Convert` si occupa di convertire da un tipo di dato ad un altro
- `ToInt32(. . .)` consente di ottenere un valore intero (a 32 bit, vedi paragrafo `4.2.1.1`)
- `temp` è l' argomento o parametro, cioè l valore da convertire
- `V =` assegna il risultato ella conversione, cioè il valore intero corrispondente all' input da tastiera ala variabile V di tipo intero

Ricorda i seguenti termini o espressioni

string	sequenza di caratteri, ricordando che ogni valore inserito da tastiera è un carattere, ad esempio Giuseppe è una stringa, ma anche 123,45 inseriti da tastiera sono una stringa
Convert.ToInt32(...)	converte il valore inserito fra parentesi tonde, di tipo stringa, in intero Convert è una classe definita nello spazio dei nomi System ToInt32 è il metodo (il modo) per ottenere un valore intero
ReadLine ( )	consente di acquisire un valore da tastiera e eventualmente di memorizzalo assegnandolo ad una variabile

3.3.1.4. Commenti

I **commenti** sono stringhe di testo incluse nel codice nel programma, che tuttavia non vengono eseguite e che servono come **documentazione** del lavoro.

Leggi il codice

```
/* esempio di commento su più righe
 * il mio primo programma in linguaggio C#
 * autore _____
 * data _____
 */

using System;
namespace informatica
{
  class esempio
  {
    static void Main()
    {
      Console.WriteLine("Buongiorno da C#!");
      Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
      Console.ReadLine();
    } //fine Main - esempio commento su una sola riga
  } //fine classe
}
```

È possibile anche utilizzare commenti del tipo

```
/// commento per generare un file  XML
```

Tali commenti possono venire estratti in modo automatico dal nostro codice e produrre un file XML di documentazione.

3.3.2. Strutture di controllo in linguaggio C#

Vengono presentate qui le principali strutture di controllo del linguaggio C# : come premessa un richiamo ai concetti di variabili e operatori, in modo da poter poi introdurre le diverse strutture attraverso il loro utilizzo in programmi di esempio completi.

3.3.2.1. Variabili e operatori

Nei primi programmi in linguaggio C# abbiamo visto alcuni esempi di utilizzo di dati: al di là dei casi particolari vengono qui presentati i tipi di dati di uso più comune; ulteriori approfondimenti nel capitolo 4.2.1.Per gestire un **dato** è necessario memorizzarlo su una **variabile**, cioè in una area della memoria **RAM**; per allocarci lo spazio per la memorizzazione è però necessario specificare preventivamente il **tipo di dato** che vorremo memorizzare, ciò al fine di consentire al computer di scegliere una tecnica di memorizzazione adeguata sia ai valori che intendiamo memorizzare (dominio), sia alle operazioni che andremo ad effettuare sui valori stessi.

In tabella i tipi di dati utilizzati in questa sezione: sono tutti tipi incorporati (in seguito vedremo come è possibile definire nuovi tipi di dati da parte del programmatore) e tipi valore, cioè le variabili corrispondenti memorizzano direttamente i valori che intendiamo associare ad esse, tranne String che è di tipo riferimento[45].

Tipo di dato	Dimensioni in memoria	descrizione	Esempio di valori	Operazioni [46]	Esempio di operazione	risultato
int	32 bit	Numero intero con segno	4 -76	aritmetiche	-5 * (2+4)	-30
					7/2	3[47]
				confronto	7 == 5	true
double	64 bit	Valori a virgola mobile, precisione 15-16 cifre	1.5643 - 789.67	aritmetiche	7.5 / 2	3.75
				confronto	4.5 < 3.5	false
char	16 bit	Carattere UNICODE	'A' '7'	confronto	'C' > 'A'	true
bool		Valori booleani	true false	confronto		
string		sequenza di zero o più caratteri Unicode	"Ciao" "C"	concatenazione Confronto ==	"b"+ "hello" "cip"=="ciop"	"bello" false

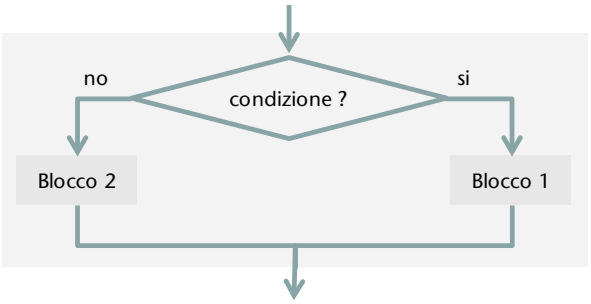
[45] In questo momento non è necessario comprendere tuti i riferimenti teorici, che saranno chiari in seguito. È importante invece sviluppare esperienza nell' uso di questi tipi di dato.

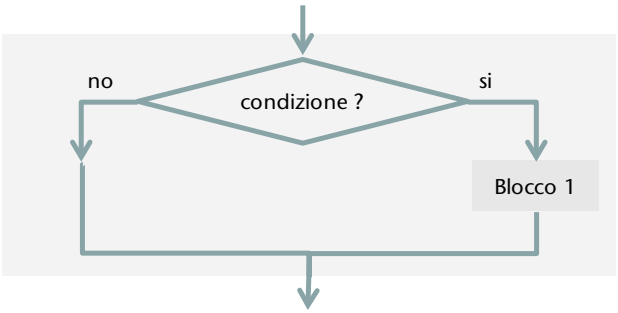
[46] Per tutti i tipi di dato presentati è definita, oltre a quelle indicate, anche l'operazione di assegazione

[47] Divisione fra numeri interi: nel risulltato non abbiamo le cifre decimali

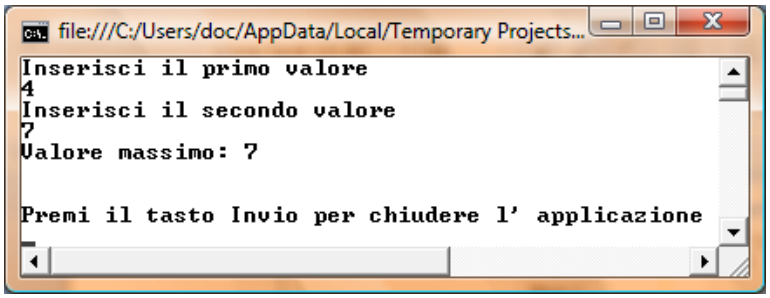
3.3.2.2. Istruzione if... else

algoritmo	Codifica

	if (condizione)
	{
	Blocco 1
	}
	else
	{
	Blocco 2
	}

	if (condizione)
	{
	Blocco 1
	}

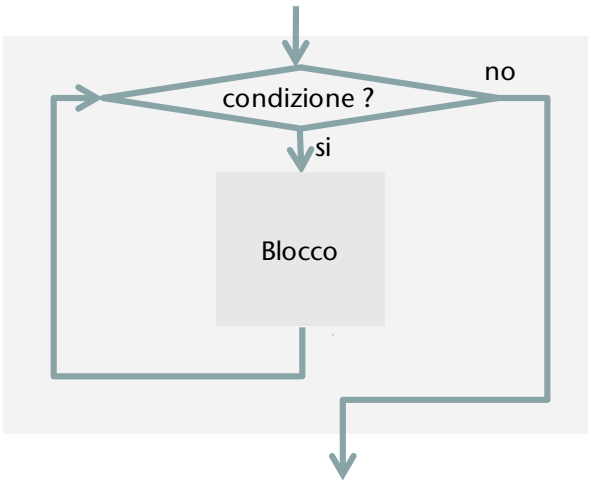
Leggi il codice corrispondente ai due algoritmi per il calcolo del massimo di due valori




```
using System;
class esempio_03
{   static void Main()
    {   int V1,V2 ;
        string temp;

Console.WriteLine("Inserisci il primo valore ");
        temp = Console.ReadLine();
        V1 = Convert.ToInt32(temp);
        Console.WriteLine("Inserisci il secondo valore ");
        temp = Console.ReadLine();
        V2 = Convert.ToInt32(temp);
        if (V1 > V2)
        {
            Console.WriteLine("Valore massimo: {0}", V1);
        }
        else
        {
            Console.WriteLine("Valore massimo: {0}", V2);
        }
        Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
        Console.ReadLine();
    }
}
```

3.3.2.3. Istruzione while

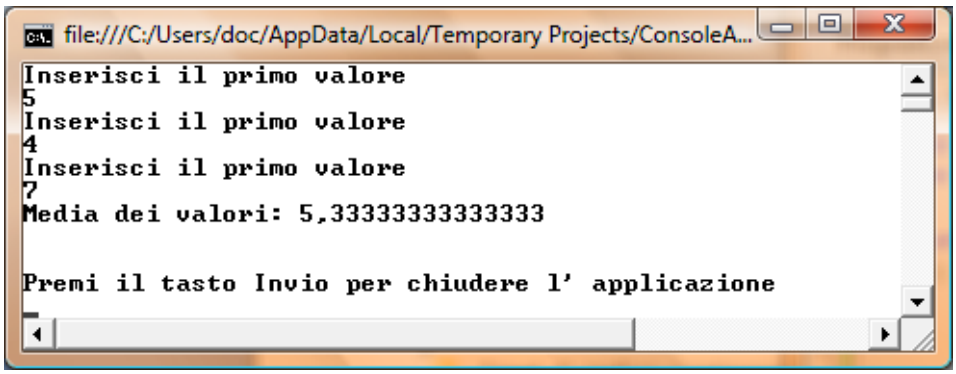
algoritmo



Codifica

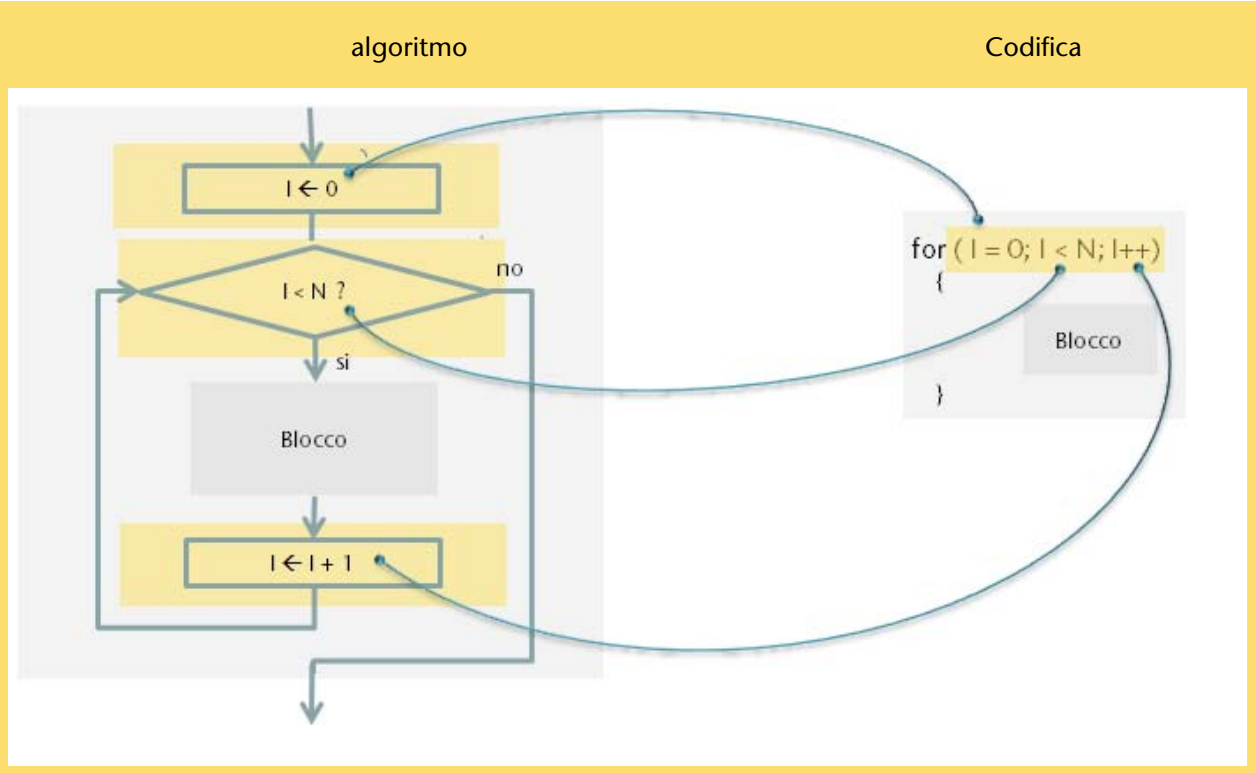
```
while ( condizione )
  {
      Blocco
  }
```

Leggi il codice corrispondente all'algorimo per calcolare la media di 3 valori (vedi paragrafo 2.3.5.3)




```
using System;
class esempio
{
   static void Main()
   {
      int V, I=0, S=0 ;
      double M;
      string temp;
      while (I < 3)
      {
         Console.WriteLine("Inserisci il primo valore ");
         temp = Console.ReadLine();
         V = Convert.ToInt32(temp);
         S = S + V;
         I = I + 1;
      }
      M = S / 3.0;
      Console.WriteLine("Media dei valori: {0}",M);

      Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
      Console.ReadLine();
   }
}
```

3.3.2.4. Istruzione for



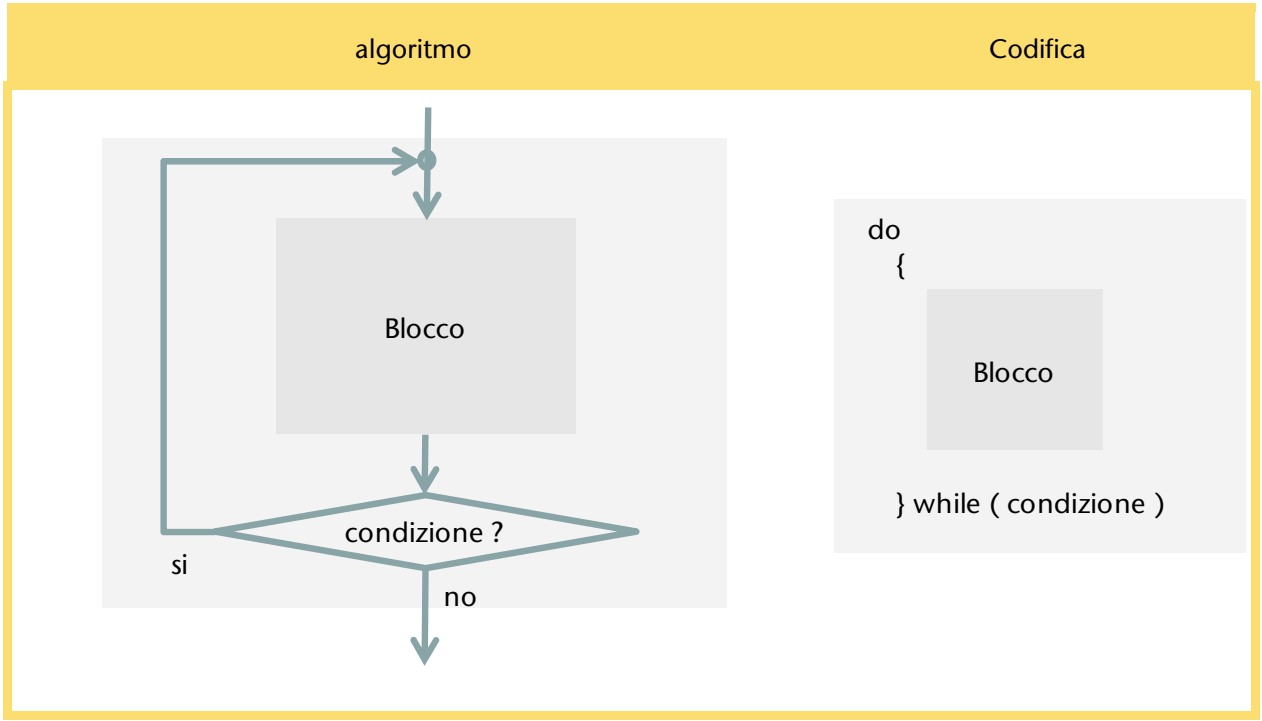
Leggi il codice corrispondente all'algoritmo per calcolare la media di 3 valori (vedi paragrafo 2.3.5.3).

Nel paragrafo precedente abbiamo proposto l'implementazione di tale algoritmo con il while, qui presentiamo la versione che utilizza il for, l' output a video è ovviamente invariato.

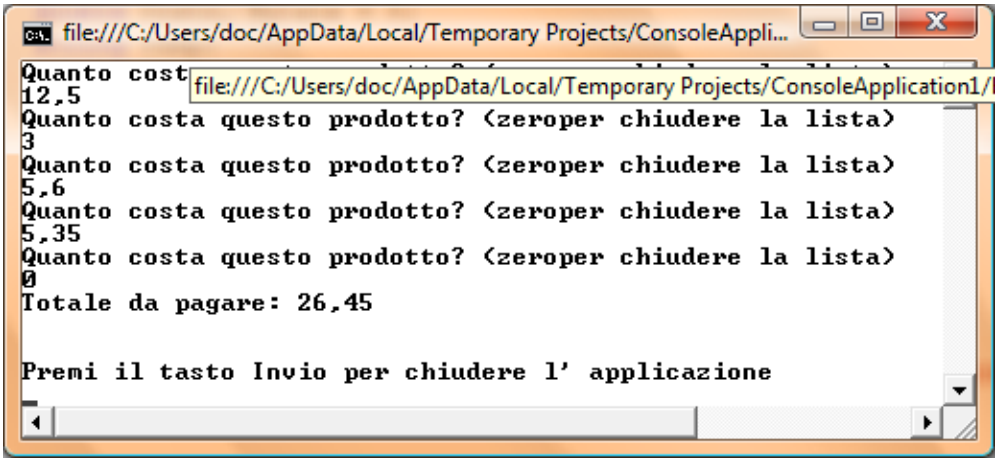
```
using System;
class esempio_03
{
   static void Main()
   {
      int V, I = 0, S = 0;
      double M;
      string temp;
      for (I=0; I<3; I++)
      {
         Console.WriteLine("Inserisci il primo valore ");
         temp = Console.ReadLine();
         V = Convert.ToInt32(temp);
         I = I + 1;
      }
      M = S / 3.0;
      Console.WriteLine("Media dei valori: {0}", M);

      Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
      Console.ReadLine();
   }
}
```

Osserva : **I = I+1** è stato scritto come **I++** (vedi operatori di incremento, paragrafo 4.2.1.7)

3.3.2.5. Istruzione do .. while



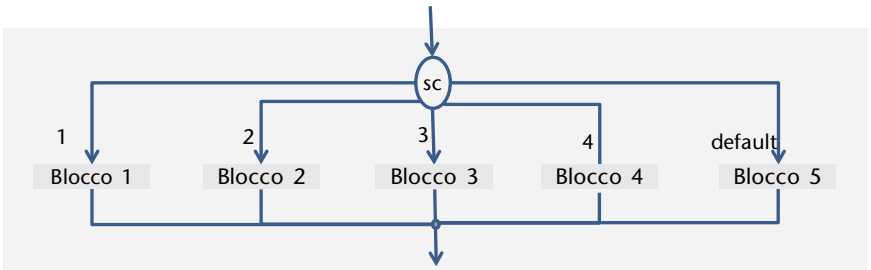
Leggi il codice corrispondente all'algoritmo per la gestione del carrello della spesa (vedi paragrafo 2.3.5.5).




```
using System;
class esempio
{
   static void Main()
   {
      double Costo, Totale = 0;
      string temp;
      do
      {
         Console.WriteLine("Quanto costa questo prodotto? (zeroper chiudere la lista) ");
         temp = Console.ReadLine();
         Costo = Convert.ToDouble(temp);
         Totale = Totale + Costo;
      } while (Costo != 0);
      Console.WriteLine("Totale da pagare: {0}", Totale);

      Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
      Console.ReadLine();
   }
}
```

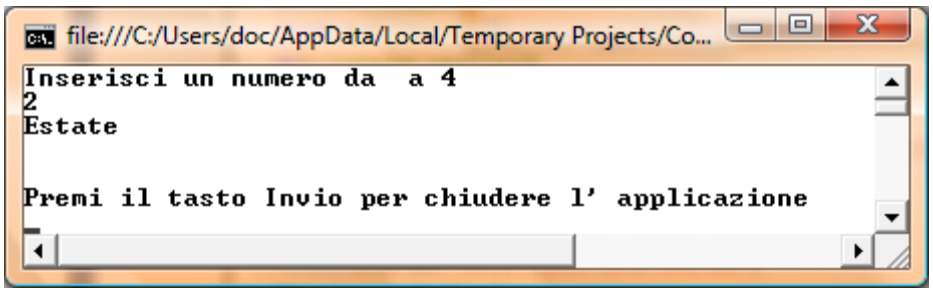
3.3.2.6. Istruzione switch

algoritmo	Codifica




```
Switch (sc)
  {
    case 1:   Blocco 1
              break;
    case 2:   Blocco 2
              break;
    case 3:   Blocco 3
              break;
    case 4:   Blocco 4
              break;
    default:  Blocco 5
  }
```

Leggi il codice corrispondente all'algoritmo per gestire i nomi delle stagioni (vedi paragrafo 2.3.3.4).



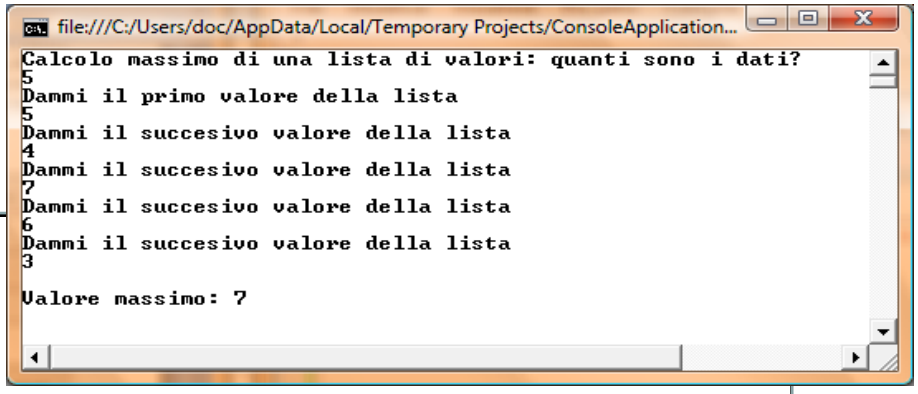

```
using System;
class esempio
{
   static void Main()
   {
      int sc;
      string temp;
      Console.WriteLine("Inserisci un numero da  a 4 ");
      temp = Console.ReadLine();
      sc = Convert.ToInt32(temp);
      switch (sc)
      {
         case 1: Console.WriteLine("Primavera "); break;
         case 2: Console.WriteLine("Estate "); break;
         case 3: Console.WriteLine("Autunno "); break;
         case 4: Console.WriteLine("Inverno "); break;
         default: Console.WriteLine("il valore inserito non è corretto "); break;
      }
      Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
      Console.ReadLine();
   }
}
```

Osserva l'istruzione **break**: si tratta di una **istruzione di spostamento** che trasferisce il controllo alla prima istruzione successiva al blocco in cui ci si trova, in queso cao alla prima istruzione che segue lo switch.

3.3.3. Sviluppo di applicazioni semplici

3.3.3.1. Massimo di N valori

Problema→ Calcolare il massimo di N valori (algoritmo: vedi paragrafo 2.3.5.4)

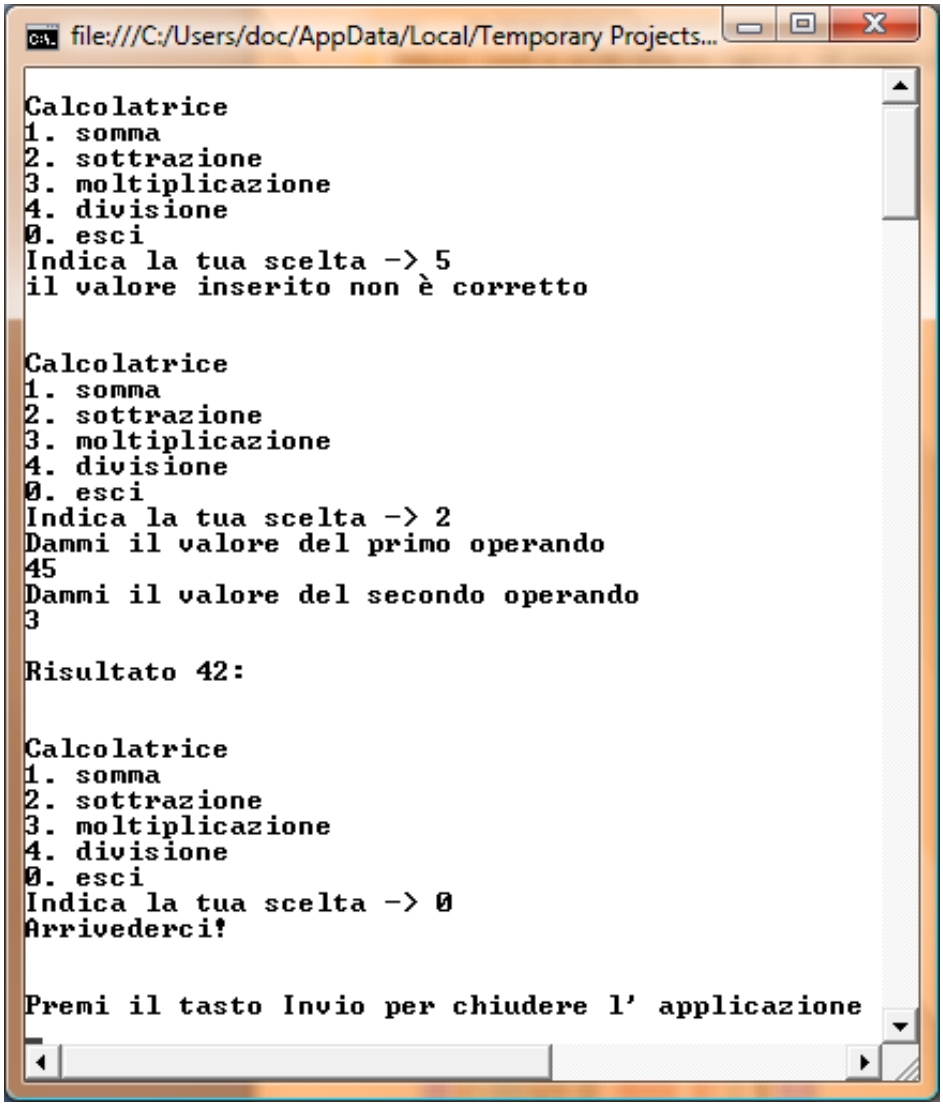



```
using System;
class Program
{
   static void Main()
   {
      int N; //numero valori da inserire
      Double V; //elenco valori da inserire
      double Max; //massimo valore dell' elenco
      int I; //contatore valori inseriti
      string tmp;

      Console.WriteLine("Calcolo massimo di una lista di valori: quanti sono i dati?");
      tmp = Console.ReadLine();
      N = Convert.ToInt32(tmp);

      Console.WriteLine("Dammi il primo valore della lista");
      tmp = Console.ReadLine();
      V = Convert.ToInt32(tmp);
      Max = V;

      //inzio il blocco ripetizione con I=1 perchè l' utente ha già specificato un valore
      for (I=1; I < N; I++)
      {
         Console.WriteLine("Dammi il succesivo valore della lista");
         tmp = Console.ReadLine();
         V = Convert.ToInt32(tmp);
         if (V > Max)
         {
            Max = V;
         }
      }
      Console.WriteLine("\nValore massimo: {0}", Max);
      Console.ReadLine();
   }
}
```

3.3.3.2. Gestione menu

Problema→ Gestire una piccola calcolatrice (algoritmo: vedi paragrafo 2.3.5.6)

```
using System;
class esempio
{
   static void Main()
   {
      int sc;
      double A=0, B=0, R=0;
      string temp;
      do
      {
         Console.WriteLine("\n\nCalcolatrice ");
         Console.WriteLine("1. somma ");
         Console.WriteLine("2. sottrazione ");
         Console.WriteLine("3. moltiplicazione ");
         Console.WriteLine("4. divisione ");
         Console.WriteLine("0. esci ");
         Console.WriteLine("Indica la tua scelta ");
         temp = Console.ReadLine();
         sc = Convert.ToInt32(temp);
         if (sc != 0)
         {
            Console.WriteLine("Dammi il valore del primo operando ");
            temp = Console.ReadLine();
            A = Convert.ToInt32(temp);
            Console.WriteLine("Dammi il valore del secondo operando ");
            temp = Console.ReadLine();
            B = Convert.ToInt32(temp);
         }
         switch (sc)
         {
            case 1: Console.WriteLine("\nRisultato {0}:", A + B);  break;
            case 2: Console.WriteLine("\nRisultato {0}:", A - B); break;
            case 3: Console.WriteLine("\nRisultato {0}:", A * B); break;
            case 4: Console.WriteLine("\nRisultato {0}:", A / B); break;
            case 0: Console.WriteLine("Arrivederci!");
                    break;
            default: Console.WriteLine("il valore inserito non è corretto ");
                    break;
         }
      } while (sc != 0);
      Console.WriteLine("\n\nPremi il tasto Invio per chiudere l' applicazione");
      Console.ReadLine();
   }
}
```



HAI IMPARATO A ...

1. Utilizzare un ambiente di sviluppo per realizare programmi
2. Scrivere ed eseguire semplici programmi in linguaggio c#
3. Conoscere ed utilizzare variabili ed operatori
4. Conoscere ed utilizzare le strutture di controllo

4. Tipi di dati

I dati che vengono memorizzati ed elaborati con il computer sono rappresentati in forma digitale, quindi la sezione inizia con la presentazione dei concetti di base relativi alla codifica delle informazioni.

Di seguito si passano a considerare le strutture dati elementari in un programma in C# che, essendo un linguaggio fortemente tipizzato, richiede che ogni variabile sia di un tipo dichiarato.

In particolare si considerano i principali tipi semplici (value type), gli array ed i file di testo.

- Rappresentazione delle informazioni e tipi di dati
- Tipi semplici (int, double, char, boolean)
- Vettori (array ad una dimensione)
- Matrici (array a due dimensioni)
- Record
- File di testo

4.1. RAPPRESENTAZIONE DELLE INFORMAZIONI E TIPI DI DATI

OBIETTIVI

Conoscere i principi di base realativi ai sistemi di codifica delle informazioni , acquisire i concetti di tipo di dato e variabile

4.1.1. Dati e informazioni

In questo capitolo si presentano i concetti di base relativi alla rappresentazione delle informazioni in formato digitale, indipendentemente dalle specificità proprie di qualsiasi linguaggio o applicazione vengano poi utilizzati per accedere ad esse. Si considerano numeri e caratteri rimandando agli approfondimenti per la codifica di immagini e suoni.

4.1.1.1. Codifica delle informazioni

Osserviamo quanto si può sperimentare con una normale lampadina: essa può essere accesa, oppure spenta. Analogamente l'hardware di un computer è costituito di circuiti elettronici per ciascuno dei quali si possono avere due distinte ma ben definite situazioni: c'è passaggio di corrente, oppure non c'è. Se a questi due diversi stati facciamo corrispondere i due valori zero (spento, off) e uno(acceso, on) possiamo capire perché ogni informazione venga codificata e memorizzata sul computer utilizzando solo due simboli[48] .

Claude E. Shannon (1916 - 2001) viene considerato il fondatore della ***teoria dell'informazione***

Problema→ l'insegnate di informatica interroga un ragazzo e si vuole registrare il risultato ottenuto, in particolare interessa sapere se la valutazione è positiva o negativa. Definire una codifica adeguata, utilizzando solo i simboli 0 e 1.

La situazione è abbastanza semplice, possiamo definire un codifica come quella proposta nella tabella seguente.

valutazione negativa (voto inferiore a 6)	0
valutazione positiva (voto 6 o superiore)	1

[48] Nel 1948 Shannon pubblicò un articolo dal titolo *The Mathematical Theory of Communication* che segnò una svolta nella storia della teoria dell'informazione. Egli scoprì ed affermò che il contenuto di informazione è in relazione al numero di unità elementari di informazioni (bit) necessarie per codificarlo. Quindi, qualunque sia la natura del messaggio - testi, suoni, immagini – esso può essere trasformato in appropriate sequenze di 1 e di 0. Ancora oggi è universalmente adottato il sistema digitale basato sui bit per la memorizzazione, l' elaborazione e la trasmissione dei dati.

Problema→ l'insegnate di informatica interroga un ragazzo e si vuole registrare il risultato ottenuto, in particolare interessa registrare l'esito della interrogazione secondo quattro livelli di giudizio (ottimo, buono, insufficiente, scarso). Definire una codifica adeguata, utilizzando solo i simboli 0 e 1.

In questo caso, se vogliamo rappresentare un insieme di valori maggiore della quantità di simboli che abbiamo a disposizione, dobbiamo pensare a soluzioni più complesse.
Facciamo ricorso alla nostra esperienza: pensiamo ai numeri, con sole 10 cifre (0, 1, 2,...9) possiamo rappresentare una grande quantità di valori. Ad esempio tutti sappiamo cosa significa il numero 123, ottenuto con una *sequenza di simboli*, ciascuno dei quali ha un significato preciso in base alla posizione.
Un sistema di codifica adeguato alla richiesta del problema potrebbe allora essere quello nella tabella a fianco, basato su sequenze di due simboli: si possono così rappresentare tutti i quattro valori richiesti.

ottimo	*Valore 11*
buono	*Valore 10*
insufficiente	*Valore 01*
scarso	*Valore 00*

Numero di bit	Potenza	Numero complessivo di valori
1	2^1	2
2	2^2	2*2 = 4
3	2^3	2*2*2 = 8
4	2^4	2*2*2*2=16
. . .	. . .	
8bit = 1 byte	2^8	2*2*2*2*2*2*2*2=256
. . .	. . .	
10	2^{10}	2*2* ..... *2 = 1024

Nei computer l'elemento base per rappresentare le informazioni è il **bit (binary digit)** che rappresenta due possibili informazioni (zero e uno): per rappresentare un maggior numero di informazioni si utilizzano *sequenze di bit*.

Spesso viene utilizzato come unità di misura il **byte** (sequenza di 8 bit).
La tabella mostra quante informazioni si possono rappresentare con N bit (devi contare il numero complessivo di valori che è possibile rappresentare, ad ogni valore corrisponde una informazione).

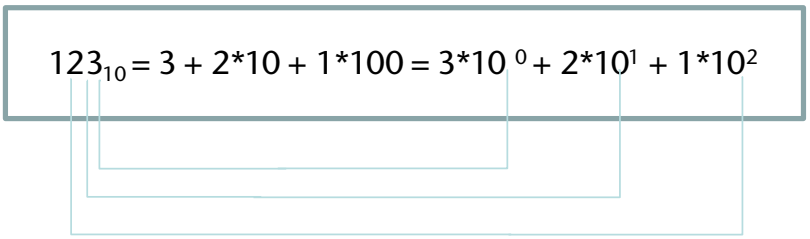
Di seguito le unità di misura utilizzate più di frequente per indicare la dimensione delle memorie.

unità di misura		Valore e potenza corrispondente		
Nome	sigla			
byte	B	1 byte = 8 bit		2^1
KiloByte	KB	1024	Circa un migliaio	2^{10}
MegaByte	MB	1 KB * 1024	Circa un milione	2^{20}
GigaByte	GB	1 MB * 1024	Circa un miliardo	2^{30}
TeraByte	TB	1 GB * 1024	Circa mille miliardi	2^{40}

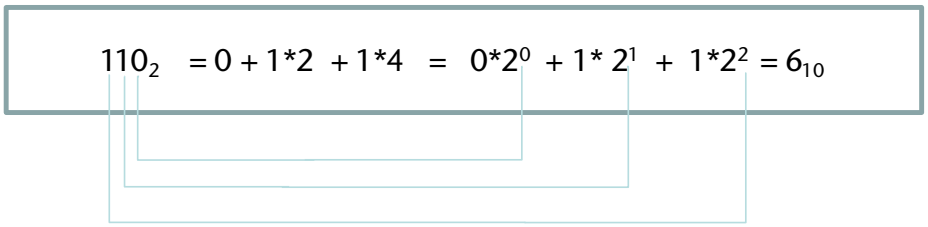
Quando si memorizzano informazioni sul computer (testo, numeri, immagini, suoni, ecc.) si ha sempre una rappresentazione digitale basata su sequenze di bit.

4.1.1.2. Dati numerici: codifica binaria

Ricordiamo le basi della rappresentazione decimale dei numeri : si tratta di una notazione posizionale (il significato delle cifre dipende dalla loro posizione, 123 è diverso da 321) basata sulle potenze di 10; osserva il seguente esempio relativo ad un numero intero[49]:



In modo del tutto analogo avviene la codifica dei numeri interi memorizzati in un computer: naturalmente i simboli da utilizzare sono solo due, quindi avremo potenze di 2, ad esempio[50]



Il **sistema binario** utilizza quindi una notazione posizionale, basata sulle **cifre zero e uno** e sulle potenze di 2.

Naturalmente è possibile calcolare il risultato di espressioni formate dalla combinazione di numeri e operatori; un esempio con valori in base dieci: 6 + 8 = 14; un altro esempio 645 + 519 = 1164. Nel primo caso effettuiamo il calcolo a memoria, nel secondo caso forse no.

Proviamo a ricordare le regole per effettuare la **somma fra due valori interi**:

riga dei riporti		1	1		
		6	4	1	+
		5	1	9	=
	1	1	6	0	

Quando calcoliamo ad esempio 1+9=10 scomponiamo il risultato in zero unità ed 1 decina (che indichiamo come riporto nella colonna delle decine).

In modo del tutto analogo si effettuano le operazioni con numeri in base due, ad esempio:

riga dei riporti		1	1		
			1	1	+
		1	0	1	=
	1	0	0	0	

Qui si comincia con 1 + 1 che dà come risultato 10_2 , cioè zero con riporto di uno

In effetti considerato che $11_2 = 1*2^0 + 1*2^1 = 1*1 + 1*2 = 3_{10}$ e che $101_2 = 1*2^0+0*2^1+1*2^2=1*1+1*4=5$ otteniamo come somma $1000_2 = 0*2^0 + 0*2^1 + 0*2^2 + 1*2^3 = 8$, quindi il risultato è corretto!
In modo analogo si effettuano tutte le altre operazioni.

---

[49] Il valore 10 mostrato in basso dopo il numero indica la base a cui ci si riferisce, in questo caso 123 in notazione decimale

[50] 110 è in base 2, cioè codifica binaria, 6 è in base dieci , codifica del sistema di numerazione decimale

Risulta molto utilizzato il **sistema esadecimale** che utilizza una notazione posizionale basata su **16 cifre** (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) e quindi sulle potenze di 16,. Per indicare la codifica esadecimale si scrive il valore preceduto da Ox, ad esempio:

$$0x352 = 352_{16} = 2*16^0 + 5*16^1 + 3*16^2 = 2 + 5*16 + 3*256 = 850_{10}$$

$$0xA3C = C*16^0 + 3*16^1 + A*16^2 = 12*16^0 + 3*16^1 + 10*16^2 = 12*1 + 3*16 + 10*256 = 2620_{10}$$

Il vantaggio di tale notazione è che rappresenta i valori in forma molto compatta rispetto ai sistemi decimale e binario, ed inoltre si presta bene ad esprimere velocemente i valori binari, in quanto la conversione fra i due sistemi è pressoché immediata, considerato che 16 è una potenza di 2, ad esempio:

1001	0110	0101	1010	0001	1110	1101	0011	1100	binario
9	6	5	A	1	E	D	3	C	esadecimale

Come si rappresentano con solo due cifre zero e uno i **numeri reali**? Innanzi tutto si esprime il numero in notazione scientifica (per questo si parla di rappresentazione in virgola mobile), quindi si rappresentano in forma binaria i due valori distinti di mantissa ed esponente. Vediamo un esempio:

$1234,567 = 0,1234567 * 10^4$ da cui Mantissa = 1234567 ed esponente = 4

4.1.1.3. Caratteri: codifica ASCII e UNICODE

Per la rappresentazione dei caratteri si utilizzano sistemi standard di codifica, cioè di corrispondenza fra caratteri e numeri interi; ad esempio alla lettera A corrisponde il valore 65, alla B il 66, e così via.

Tipologia di codifica	Numero bit	Descrizione
ASCII standard	7 bit	Alfabeto anglosassone, rappresenta 128 caratteri
ASCII esteso	8 bit	Rappresenta 256 caratteri, però cambia con la lingua che si utilizza
UNICODE	16 bit	originariamente a 16 bit (quattro cifre esadecimali) rappresenta 65.536 caratteri

Vengono rappresentati con questa codifica anche altri caratteri, quali le parentesi, i segni di interpunzione, le cifre numeriche (intese però non come numeri su cui effettuare calcoli), ed altri ancora (ad esempio lo spazio,codice 32, il carattere di fine riga che si indica con 'CR', codice 13). Nello stesso modo si rappresentano le parole, che non sono altro se non sequenze di caratteri , dette anche stringhe .

Un tempo era molto diffusa la **codifica ASCII**[51], in grado di codificare però, anche nella sua versione estesa, solo 256 caratteri; attualmente si fa riferimento in genere alla **codifica Unicode**[52].

La codifica Unicode attribuisce un numero univoco a ogni carattere, indipendentemente dalla piattaforma, dall'applicativo, dalla lingua; per i caratteri che erano già codificati in ASCII viene mantenuta la stessa codifica, così ad esempio A ha il codice ASCII 65_{10} e conserva con la codifica Unicode lo stesso codice, che viene rappresentato in forma esadecimale come $U+41_{16}$ (il prefisso U+ indica Unicode). La tabella completa della codifica dei caratteri latini si trova in http://www.unicode.org/charts/PDF/Uoooo.pdf .

[51] Acronimo di American Standard Code for Information Interchange

[52] Vedi http://www.unicode.org/standard/translations/italian.html

4.1.2. Tipi di dati in C#

Si precisa il significato di alcuni termini di interesse generale e si mostra un esempio di memorizzazione di dato con riferimento al linguaggio C#.

4.1.2.1. Informazione, dato e tipo di dato

Si è visto che in un computer tutte le informazioni sono codificate in forma binaria, cioè come sequenza di zero e uno.

Consideriamo ad esempio il valore 5: esso come può essere rappresentato?

Non c'è una risposta univoca, perché 5 è un **dato**, ma non sappiamo a quale **informazione** corrisponda.

Per esempio posso dire: "questo oggetto costa 15 Euro", oppure "Luigi abita in via Roma, 15" . I due diversi contesti mi danno indicazioni differenti: nel primo caso 15 rappresenta un numero, può essere opportuno codificarlo come un intero, in modo da poter poi eventualmente effettuare delle operazioni matematiche, mentre nel secondo caso 15 fa parte di un indirizzo, quindi posso considerarlo all' interno di un testo, per cui la codifica sarà quella UNICODE.
Vediamo quindi che è importante considerare non solo i **valori**, ma anche le **operazioni** che su tali valori vogliamo effettuare: a tal proposito si parla di **tipo di dato**.

4.1.2.2. Variabili e tipi di dato in C#

Gli esempi di programmi in linguaggio C# del capitolo precedente hanno mostrato che, quando si utilizza una variabile per memorizzare un certo valore, è sempre necessario specificare il tipo di dato corrispondente. Si dice a tal proposito che **C# è un linguaggio fortemente tipizzato**, ogni variabile, ogni costante dispone di un tipo ed ogni espressione che restituisce un valore.

Nel **linguaggio C#** distinguiamo due categorie[53] di tipi di dato:

- **tipi valore**: le variabili di questi tipi memorizzano direttamente i valori
- **tipi riferimento**: le variabili memorizzano solo un riferimento alla area di memoria in cui sono effettivamente memorizzati i dati. I tipi riferimento corrispondono agli oggetti.

Boxing e unboxing
È possibile convertire un tipo di valore in un tipo di riferimento e nuovamente in un tipo di valore tramite boxing e unboxing. A eccezione dei tipi di valore boxed, non è possibile convertire un tipo di riferimento in un tipo di valore.

[53] In un contesto unsafe sono disponibili anche i tipi puntatore, vedi per approfondimenti il manuale MSDN

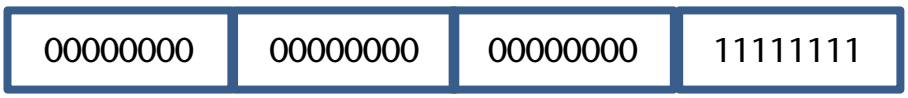
4.1.2.3. Organizzazione della memoria di un calcolatore

Una **variabile** è una area di memoria cui vengono associati: identificatore, tipo, indirizzo, valore.

Riprendiamo l' esempio del paragrafo precedente e vediamo come può essere memorizzato il valore 15.

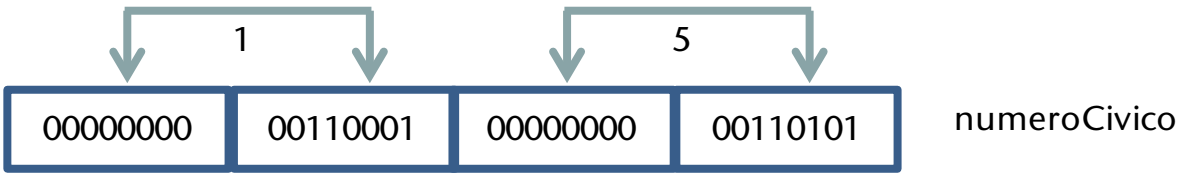
```
using System;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main( )
      {
         int costo = 15;
         string NumeroCivico = "15";
      }
   }
}
```

Vediamo con precisione cosa abbiamo memorizzato un consiglio: esegui il codice visualizzando la scheda delle variabili locali per verificare direttamente ciò che accade: con Visual Studio utilizza F11 per l' esecuzione passo-passo, con Sharpdevelper usa i breakpoint)



Dimensione:4 byte

Tipo di dato: intero – codifica binaria



Dimensione: 4 byte

Tipo di dato: stringa – codifica UNICODE

4.2. TIPI SEMPLICI

PREREQUISITI

[Concetto di tipo di dato e di variabile]

OBIETTIVI

[Conoscere i principali tipi di dati semplici e utilizzarli per la soluzione di problemi]

4.2.1. Tipi di dati semplici (value type)

*Presentiamo alcuni dei più utilizzati tipi di dati incorporati del linguaggio C#, che corrispondono ai tipi predefiniti nello spazio dei nomi System (e quindi nel .NET Framework); di seguito gli operatori che agiscono sui diversi tipi di dato. Il tipo record, pur essendo value type, viene considerato in seguito (vedi **4.5.1**) mentre viene inserito qui il tipo string, anche se è un reference type, per i motivi spiegati nel paragrafo 4.2.1.4.*

4.2.1.1. int

Tipo di dato C#	Tipo .NET Framework	Dimensioni	Intervallo valori
int	System.Int32	32 bit	Numero intero con segno da -2,147,483,648 a 2,147,483,647

Sono definiti anche altri tipi di dati per rappresentare numeri interi, con o senza segno, utilizzando da un minimo di 8 sino a 64 bit[54].

4.2.1.2. double

Tipo di dato C#	Tipo .NET Framework	Dimensioni	Intervallo valori
double	System.Double	64 bit	Valori a virgola mobile da $\pm 5.0 \times 10^{-324}$ a $\pm 1,7 \times 10^{308}$ con precisione di 15-16 cifre

[54] Vedi manuale MSDN, tipi uint (senza segno 32 bit), byte (senza segno 8 bit), short (senza segno 16 bit), ushort (senza segno 16 bit), long (con segno 64 bit), ulong (senza segno 64 bit).

Leggi il codice

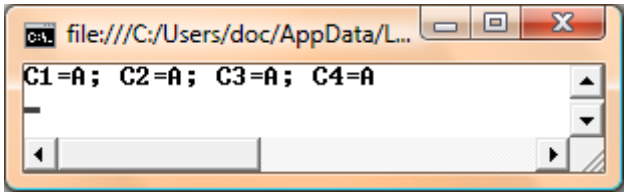
```
1  using System;
2  class Program
3    {static void Main( )
4      {
5        int i = 15, k;
6        double d, e;
7      1 k = i / 2;     //k vale 7
8      2 d = i / 2;     //d vale 7.0
9      3 e = i / 2.0;   //e vale 7.5
10       Console.ReadLine();
12     }
13   }
```

1	Divisione fra interi, il risultato 7 viene assegnato ad un intero
2	Divisione fra interi, il risultato 7 viene convertito implicitamente in double 7.0 ed assegnato ad una variabile di tipo double
3	Divisione fra un intero i che vale 15 ed un double 2.0: l'intero viene convertito prima della divisione, quindi si ha una divisione fra double, il risultato è 7.5

4.2.1.3. char

Tipo di dato C#	Tipo .NET Framework	Dimensioni	Intervallo valori
char	System.Char	16 bit	Carattere Unicode Da U+0000 a U+ffff

Osserva e leggi il codice corrispondente




```
1  using System;
2  class Program
3    {
4      static void Main( )
5      {
6        char C1 = 'A';       // carattere letterale
7        char C2 = '\x0041';  // valore esadecimale
8        char C3 = '\u0041';  // valore Unicode
9        char C4 = (char)65;  // Cast (conversione esplicita) da un valore intero
10       Console.WriteLine("C1={0}; C2={1}; C3={2}; C4={3}",C1,C2,C3,C4);
11       Console.ReadLine();
12     }
13   }
```

Una **sequenza di escape** è una successione di 2 caratteri che provocano particolari comportamenti del sistema: il primo dei caratteri è il backslash \ che serve ad introdurre il secondo il quale indica in modo specifico l'operazione da effettuare: di seguito si indicano alcuni dei casi più comuni.

campanello	\a
backspace → torna indietro di una posizione	\b
newline → va a capo	**\n**
tabulazione orizzontale	\t

Osserva e leggi il codice corrispondente


```
1  using System;
2  class Program
3  { static void Main()
4     {
5        Console.WriteLine("elenco");
6        Console.WriteLine("\n\nnomi\tvoti");
7        Console.WriteLine("luigi\t6");
8        Console.WriteLine("lia\t8");
9        Console.WriteLine("ugo\t7");
10       Console.WriteLine('\a');
11       Console.ReadLine();
12    }
13 }
```

4.2.1.4. string

Una **stringa** è una sequenza di zero o più caratteri Unicode.

Osserva e leggi il codice corrispondente


```
using System;
class Program
{
   static void Main()
   {
      string a = "Ciao";
      string nome = "Matteo";
      string saluto = "buongiono\na tutti"; //osserva il carattere di escape \n
      string percorso = @"c:\miaCratella\nuovofile.txt"; //@ indica che le sequenze di escape \n
                                 //NON deve essere elaborata
      Console.WriteLine(a);
      Console.WriteLine(nome[0]); // visualizza la prima lettera del nome
      Console.WriteLine(nome[1]); // visualizza la seconda lettera del nome
      Console.WriteLine(nome[2]); // visualizza la terza lettera del nome
      Console.WriteLine(saluto);
      Console.WriteLine(percorso);
      Console.ReadLine();
   }
}
```

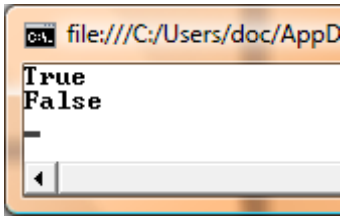
Si presentano qui le stringhe (vedi 4.3.1.7), anche se non sono dati di tipo semplice, per due motivi:

- si tratta di un dato di uso veramente comune, quindi si preferisce presentarlo in questo primo elenco;
- anche se si tratta di un tipo riferimento si comporta spesso come se fosse un tipo valore: questa scelta da parte dei progettisti del linguaggio è motivata dal fatto che in questo modo se ne rende più agevole l'utilizzo (vedi di seguito in particolare gli operatori di uguaglianza == e di disuguaglianza != che confrontano i valori, non i riferimenti).

4.2.1.5. Boolean

Tipo di dato C#	Tipo .NET Framework	Dimensioni	Intervallo valori
bool	System.Boolean		True, false

Osserva e leggi il codice corrispondente




```
1  using System;
2  class Program
3  {
4     static void Main()
5     {
6        bool x = true;
7        bool y;
8        Console.WriteLine(x);
9        y = 6 < 5;
10       Console.WriteLine(y);
11       Console.ReadLine();
12    }
13 }
```

4.2.1.6. Cast dei tipi

Ad ogni **tipo di dato** corrisponde un insieme di valori ed un insieme di operazioni: è possibile convertire un dato da un tipo ad un altro.
Tale conversione avviene implicitamente quando non si ha perdita di informazione (ad esempio il passaggio da un intero ad un double), in caso contrario deve essere esplicitamente richiesta dal programmatore con un **cast** che richiama l'operatore di conversione.

A ← Da	Tipo conversione	esempio
double ← int	implicita	double x=123;
int ← double	esplicita (cast)	int y = (int) 3.7; //y vale 3
int ← char	implicita	int i = 'A'; // i vale 65
char← int	esplicita (cast)	char Carattere = (char)65;

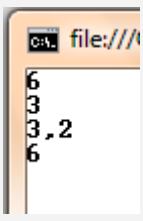
4.2.1.7. Operatori

La tabella di riepilogo mostra gli operatori utilizzati in questo volume, quindi per ciascuno di essi seguono descrizione dettagliata ed esempi di utilizzo.

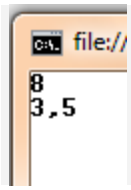
operatore	simbolo	descrizione
Op. di assegnazione	=	Assegna un valore.
Op. matematici	+, - ,* ,/ ,%	Addizione, sottrazione, Moltiplicazione, divisione, modulo
Op. concatenazione	+	concatena due stringhe o stringhe e tipi numerici
Operatori di incremento e decremento	v ++, ++v, v--, --v	Incrementa (decrementa) la variabile v di un'unità.
Op. assegnazione composta	v += n, v -= n, v *= n, v/=n;	Incrementa (decrementa) la variabile v di n unità, Moltiplica (divide) la variabile v per n unità
Op. relazionali	==, !=, <, <=, >, >=	confronta due valori
Op. logici	&&, ||, !	AND, OR, NOT
operatori primari	vedi di seguito schema in dettaglio	indichiamo qui operatori importanti trattati nel testo, per un elenco completo vedi manuale MSDN

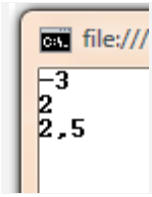
operatore di assegnazione	Esempi
= **assegnazione**	**Calcola il valore che risulta dalla espressione a sinistra del simbolo di uguale (left value) e lo assegna alla variabile indicata a destra (right value)**

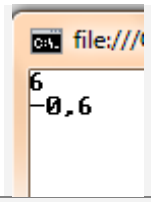
```
   static void Main()
{
   int a=5,b;
   double x=3.2,y;
   a = a + 1;  // assegnazione fra interi
   b = (int)x; // conversione esplicita da double a intero
   Console.WriteLine(a);
   Console.WriteLine(b);

   y = a;      // conversione implicita da intero a double
   Console.WriteLine(x);
   Console.WriteLine(y);
}
```

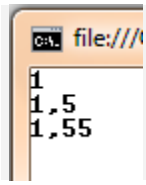


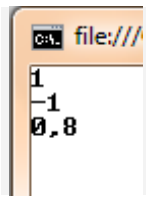
Operatori matematici	Esempi

+
Somma
	Somma fra tipi numerici anche differenti->risultato con conversione automatica
	`static void Main()`
	`{`
	`  Console.WriteLine(3 + 5);  // somma fra interi`
	`  Console.WriteLine(3 + .5);  // somma intero + double`
	`}`
	 

-
sottrazione
	Sottrazione fra tipi numerici anche differenti, ->risultato con conversione automatica
	`static void Main()`
	`{ int a = 3;`
	`  Console.WriteLine(- a);  // operatore unario`
	`  Console.WriteLine(a - 1);  // sottrazione fra interi`
	`  Console.WriteLine(a - .5);  // sottrazione fra intero e double`
	`}`
	 

*****
moltiplicazione
	prodotto fra tipi numerici anche differenti, ->risultato con conversione automatica
	`static void Main()`
	`{`
	`  Console.WriteLine(3 * 2);  //intero * intero`
	`  Console.WriteLine(-3 * .2); //intero * double}`
	`}`
	 

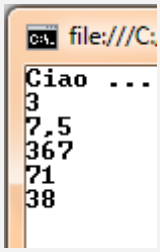
/
divisione
	Divisione fra interi → risultato intero
	Divisione fra tipi numerici differenti ->risultato con conversione automatica
	`static void Main()`
	`{  Console.WriteLine(3 / 2);  //risultato intero`
	`  Console.WriteLine(3 / 2.0);//risultato double`
	`  Console.WriteLine(3.1 / 2);//risultato double`
	`}`
	 

%
modulo
	Calcola il resto della divisione
	`static void Main()`
	`{  Console.WriteLine(3 % 2);  // risultato intero`
	`  Console.WriteLine(-3 % 2);  // risultato intero`
	`  Console.WriteLine(3.0 % 2.2); // risultato double`
	`}`
	 

Precedenza degli operatori aritmetici	**Vengono utilizzate le normali regole definite in matematica**
	`static void Main()`
	`{  int x = 1 + 3 * 100;  // prima la moltiplicazione`
	`  int y = (1 + 3) * 100;  // prima la somma`
	`  Console.WriteLine(x);`
	`  Console.WriteLine(y);  }`
	 

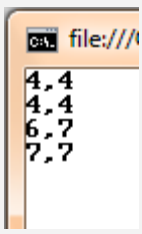
Operatore di concatenazione	Esempi
+ **concatenazione** **somma**	**Concatenazione fra stringhe, caratteri e tipi numerici, con conversione automatica dei numeri in stringhe.** Sommando un valore numerico con un carattere quest'ultimo viene trasformato in numero ed eseguita la somma: il numero è dato dal codice UNICODE del carattere stesso.

```
static void Main()
{
string a = "Ciao";
string b = " ...";
string c;
c = a + b;            // concatenazione di stringhe
Console.WriteLine(c);

Console.WriteLine(+3);            // valore intero positivo
Console.WriteLine(3+4.5);         // somma fra valori numerici
Console.WriteLine('3' + "67"); // concatena carattere e stringa
Console.WriteLine('A' + 6);       // somma di A (65) e intero
Console.WriteLine(3.0 + "8"); // concatenazione di stringhe
}
```

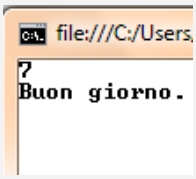


Operatori di incremento e decremento	Esempi
++ **incremento** **incrementa il proprio operando di 1** **--** **decremento** **decrementa il proprio operando di 1** **Opera in modo analogo all' operatore ++**	**Forma prefissa→ incrementa , poi esegue eventuali altre operazioni** **Forma postfissa → esegue eventuali operazioni, di seguito incrementa**

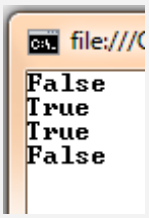
```
static void Main()
{
   double d;
   d = 3.4;
   Console.WriteLine(++d); //prima incrementa, poi stampa il nuovo valore
   Console.WriteLine(d);
   d = 6.7;
   Console.WriteLine(d++); //prima stampa il vecchio valore, poi incrementa
   Console.WriteLine(d);
}
```



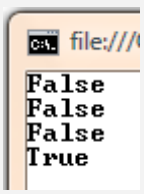
Operatori di assegnazioni composte	Esempi

+=
-=
***=**
/=
%=
 	
assegnazione di somma
sottrazione
moltiplicazione
divisione
resto
 	
concatenazione
	x += y equivale a **x = x + y**
	x -= y equivale a **x = x - y**
	x *= y equivale a **x = x * y**
	x /= y equivale a **x = x / y**
	x %= y equivale a **x = x % y**
	nella forma contratta la x viene valutato una sola volta
	L'operatore += viene inoltre utilizzato per specificare un metodo chiamato in risposta ad un evento (vedi paragrafo Gestione eventi e MessageBox)
	`static void Main()`
	`{ int a = 3;`
	`a += 4; //somma`
	`Console.WriteLine(a);`
	`string s = "Buon ";`
	`s += "giorno."; //concatenazione`
	`Console.WriteLine(s);`
	`}`
	 

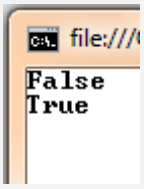
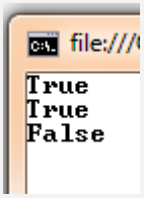
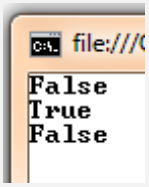
Operatori di Relazione	Esempi

==
uguaglianza
	tipi di valore predefiniti → restituisce true se i valori degli operandi sono uguali e false in caso contrario
	tipo stringa → confronta i valori delle stringhe
	`static void Main()`
	`{ Console.WriteLine((2 + 3) == 4);`
	`Console.WriteLine((2 + 3) == 5);`
	`string a = "salve";`
	`string b = "salve";`
	`string c = "ciao";`
	`Console.WriteLine(a == b);`
	`Console.WriteLine(a == c);`
	`}`
	 

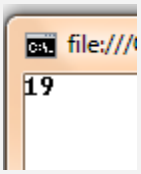
!= **disuguaglianza**	**Non uguaglianza**, è l'opposto del caso precedente

>
Maggiore
 	
>=
Maggiore o uguale
	operatore relazionale che restituisce **true** se il primo operando è maggiore (oppure maggiore o uguale) del secondo e **false** in caso contrario.
	`static void Main()`
	`{ Console.WriteLine(1 > 1.1);`
	`Console.WriteLine(1.1 > 1.1);`
	`Console.WriteLine(1 >= 1.1);`
	`Console.WriteLine(1.1 >= 1.1);`
	`}`
	 

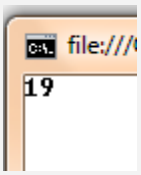
< **Minore** **<=** **Minore o uguale**	restituisce **true** se il primo operando è minore (oppure minore o uguale) del secondo e **false** in caso contrario.

<table>
<tr><th>Operatori logici</th><th>Esempi</th></tr>
<tr><td>&&
AND condizionale
<table><tr><td>R</td><td>S</td><td>R && S</td></tr><tr><td>vero</td><td>vero</td><td>vero</td></tr><tr><td>vero</td><td>falso</td><td>falso</td></tr><tr><td>falso</td><td>vero</td><td>falso</td></tr><tr><td>falso</td><td>falso</td><td>falso</td></tr></table></td><td>Verifica se entrambe le condizioni sono soddisfatte valutando il secondo operando solo se necessario (cioè solo se la prima condizione è verificata)
<pre>
static void Main()
{
  int a=3, b=4;
  bool R, S;
  R = (a<2) && (b<5);   //falso && vero → falso
  S = (a < 20) && (b < 5); //vero && vero → vero
  Console.WriteLine(R);
  Console.WriteLine(S);
}
</pre>  </td></tr>
<tr><td>||
OR condizionale
<table><tr><td>R</td><td>S</td><td>R || S</td></tr><tr><td>vero</td><td>vero</td><td>vero</td></tr><tr><td>vero</td><td>falso</td><td>vero</td></tr><tr><td>falso</td><td>vero</td><td>vero</td></tr><tr><td>falso</td><td>falso</td><td>falso</td></tr></table></td><td>Verifica se almeno una delle due condizioni è soddisfatta valutando il secondo operando solo se necessario
<pre>
static void Main()
{
int a=3, b=4;
bool R, S, T;
R = (a<2) || (b<5);   //falso oppure vero --> vero, perchè
almeno una delle due condizioni è verificata
S = (a < 20) || (b < 5); //vero oppure vero --> vero
T = (a < 2) || (b > 5); //falso oppure falso --> falso, tutto falso
                         //neanche una condizione è vera
Console.WriteLine(R);
Console.WriteLine(S);
Console.WriteLine(T);
}
</pre>  </td></tr>
<tr><td>!
negazione
<table><tr><td>R</td><td>! R</td></tr><tr><td>vero</td><td>falso</td></tr><tr><td>falso</td><td>vero</td></tr></table></td><td>operatore unario che nega l'operando, che deve essere una espressione booleana
<pre>
static void Main()
{
  Console.WriteLine(!true);
  Console.WriteLine(!false);
  Console.WriteLine(!(3>2));
}
</pre>  </td></tr>
<tr><td>? :
condizionale</td><td>Si tratta di una forma concisa di if ...else che si scrive come
(condizione) ? primaEspressione : secondaEspressione;
Se la condizione è vera si esegue la prima espressine, altrimenti la seconda
<pre>
static void Main()
{
  int voto = 7;
  string giudizio;
  giudizio = (voto > 6) ? "bravo" : "studia";
  Console.WriteLine(giudizio);
}
</pre>  </td></tr>
</table>

Operatori primari	Esempi
operatore punto	**Accesso ad un membro** specifica un membro di un tipo di dato (campo di un record, librerie di classi di .NET Framework o di classi definite dall'utente) o di uno spazio dei nomi
	System.Console.WriteLine("hello"); MessageBox.Show("ciao"); Alunno.Voto = 7;
( ) **parentesi tonde**	**specifica l'ordine in cui devono essere eseguite le operazioni di un'espressione**

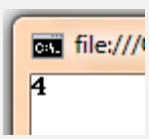
	static void Main()
	{
	int a;
	a = 5 + (2 * (6 + 1));
	Console.WriteLine(a);
	}
	 

( ) **parentesi tonde**	**conversioni di tipi (cast)** vedi paragrafo 4.2.1.6
( ) **parentesi tonde**	**specifica l'ordine in cui devono essere eseguite le operazioni di un'espressione**

	static void Main()
	{
	int a;
	a = 5 + (2 * (6 + 1));
	Console.WriteLine(a);
	}
	 

[ ]
parentesi quadre
	Gestione dati di tipo array
	static void Main()
	{
	int [ ] a = { 3, 5, 7 };
	Console.WriteLine(a[2]);
	}
	 

new	**Creazione di oggetti** Int [ ] = new int [3]; Label lblRisultato; lbl = new Label();

sizeof	**dimensione in byte di un tipo valore**
	static void Main()
	{
	int dimensione;
	dimensione= sizeof(int);
	Console.WriteLine(dimensione);
	}
	 

4.3. VETTORI (ARRAY AD UNA DIMENSIONE)

PREREQUISITI

Conoscere i tipi di dati semplici]

OBIETTIVI

Conoscere il tipo di dati array ad una dimensione ed utilizzarlo in modo adeguato per la soluzione di problemi

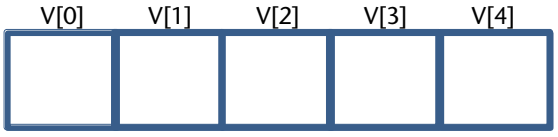
4.3.1. Dichiarazione ed inizializzazione di vettori

4.3.1.1. Un problema da risolvere

Problema→ Oggi vengono interrogati 5 alunni, a ciascuno di essi viene assegnato un voto (numero intero fra 1 e 10): interessa conoscere la media dei voti ed il numero di alunni che ottengono una valutazione superiore alla media.

Il problema da risolvere sembra abbastanza semplice ma, se pensiamo agli esempi visti fin qui, nasconde una difficoltà: chiedendo i dati ad uno ad uno ed elaborandoli per calcolare la media non possiamo rispondere al secondo quesito proposto, in quanti i valori non sono più memorizzati.
Si potrebbe pensare di chiedere all'utente di fornire una seconda volta l'elenco dei voti (in modo da poter effettuare il confronto fra ciascun voto ed il valor medio, ora che lo conosciamo), ma questa evidentemente non è una buona soluzione!
Un'altra possibilità sarebbe quella di prevedere una variabile per ciascun voto, ma in questo modo non potremmo risolvere il problema in modo generale, e la soluzione non sarebbe comunque molto agevole da gestire se il numero di voti dovesse aumentare (ad esempio: 25 alunni, 25 variabili solo per i voti): prova a pensare all'algoritmo corrispondente!

La difficoltà non è dovuta all'algoritmo, ma alla gestione dei dati. Fin qui abbiamo utilizzato variabili di tipo semplice, che possono memorizzare un solo valore per volta: è importante imparare a conoscere variabili di tipo strutturato, che possono memorizzare contemporaneamente diversi valori.



Un **vettore** è una variabile costituita da elementi tutti dello stesso tipo (ad esempio interi), ciascuno dei quali può memorizzare un valore. Una variabile di tipo vettore è identificata dal nome (ad esempio V); si individuano i diversi **elementi** dalla loro **posizione** all'interno della struttura, indicata con un numero racchiuso fra parentesi quadre, contando a partire dalla posizione zero.
Nel caso del problema proposto sarà opportuno utilizzare un vettore di interi di 5 elementi, qualora gli alunni dovessero essere 25 non sarà necessario modificare la soluzione, ma si potrà semplicemente utilizzare un vettore di 25 elementi[55].

[55] In figura i quadrati rappresentano gli elementi del vettore, sopra a ciascuno di essi il proprio identificatore costituito dal nome del vettore e dall' indice del singolo elemento

Analisi

Chiediamo di conoscere i voti attribuiti a ciascun alunno, memorizzando tutti i valori ottenuti, e calcolandone la media. Successivamente si confronta ciascun voto con la media calcolata al fine i conteggiare il numero di studenti con valutazione superiore alla media.

Tabella di descrizione delle variabili

Nome	Descrizione	Tipo di dato	Input/Output/lavoro
V[5]	Voti ottenuti dagli studenti	Vettore di interi	Input
I	Contatore che individua i singoli studenti e quindi la posizione degli elementi corrispondenti del vettore	intero	lavoro
S	Somma dei voti	Intero	lavoro
M	Media dei voti	double	Output
B	Contatore numero studenti Bravi, con voto superiore a M	intero	output

Algoritmo

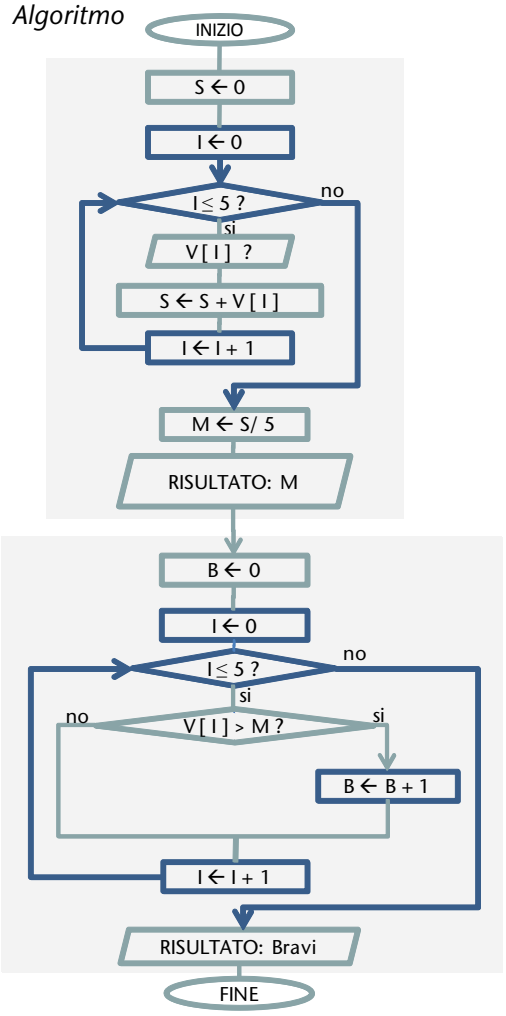


Tabella di traccia[56]

I	S	M	V[0]	V[1]	V[2]	V[3]	V[4]	B	Output
0	0								
			7						
1 [57]	7								
2	11			4					
3	17				6				
4	25					8			
	31						6		
5									
		6.2							6.2
0								1	
1									
2									
3								2	
4									
5									2

[56] Ad ogni elemento del vettore corrisponde una colonna

[57] Per brevità sono talvolta indicate nella stessa riga tutte le operazioni del blocco ripetizione

4.3.1.2. Dichiarazione di un vettore

Un **vettore** (array ad una dimensione) è una **collezione di dati dello stesso tipo, ciascuno identificato dalla posizione**. Di seguito un esempio di vettore di 5 elementi di tipo intero.

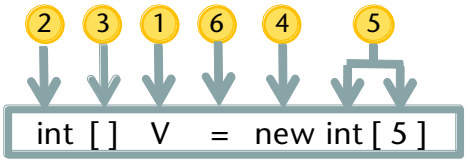


Figura 4-2 - dichiarazione vettore di 5 interi

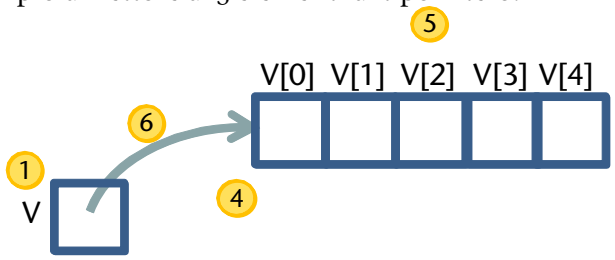


Figura 4-1 rappresentazione grafica della memoria RAM con vettore di 5 interi

1	V	Identificatore, cioè nome del vettore
2	Int	Tipo di dato degli elementi del vettore
3	[ ]	Operatore di indicizzazione: mostra che V è un vettore
4	New	Operatore: alloca lo spazio per gli elementi del vettore
5	int [5]	Indica che il vettore è costituito di 5 elementi di tipo intero
6	=	Operatore: assegna a V il riferimento agli elementi allocati in memoria

Un vettore è un oggetto
Tenicamente un vettore è un oggetto derivato dalla classe astratta System.Array

Un **vettore** è una **variabile di tipo riferimento**, in quanto nella variabile V non vengono memorizzati i valori, ma solo il riferimento (puntatore) all'area di memoria che viene allocata con l'operatore new ed in cui saranno effettivamente memorizzati i dati.

4.3.1.3. Inizializzazione di un vettore

È possibile inizializzare un array, cioè specificare i valori al momento stesso della creazione.

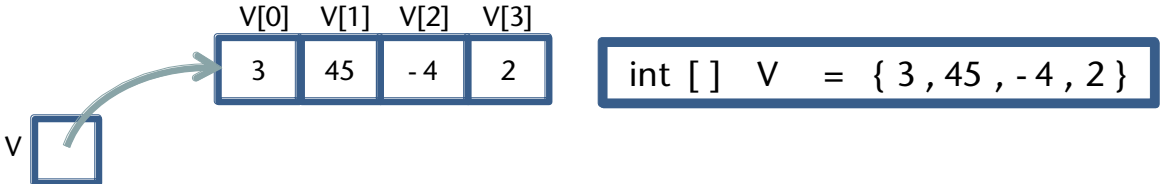
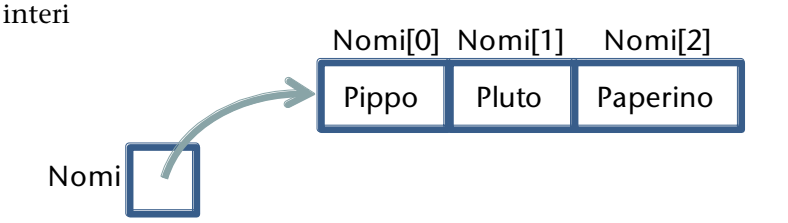


Figura 4-3 inizializzazione vettore di interi



Osserva bene
La rappresentazione grafica del vettore di stringhe proposta è una semplificazione!

String [ ] Nomi = { "Pippo" , "Pluto" , "Paperino" }

Figura 4-4 inizializzazione vettore di stringhe

4.3.1.4. Assegnazione valori agli elementi di un vettore

Osserva e leggi il codice corrispondente

Osserva il codice con la dichiarazione del vettore e le assegnazioni di valori agli elementi nelle posizioni zero, 1 e 2 e verifica il risultato nella finestra delle variabili locali durante l'esecuzione del programma avviata con F11 .

```
using System;

namespace ConsoleApplication1ebook
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] V = new int[5];
            V[0] = 10;
            V[1] = 3;
            V[2] = V[1] + 2;
        }
    }
}
```

Variabili locali

Nome	Valore	Tipo
args	{string[0]}	string[]
V	{int[5]}	int[]
[0]	10	int
[1]	3	int
[2]	5	int
[3]	0	int
[4]	0	int

Osserva che si accede agli elementi del vettore mediante nome del vettore, indice (valore che indica la posizione dell'elemento) e operatore di indicizzazione [ ]

4.3.1.5. Contatore come indice degli elementi di un vettore

Assegnare ai 5 elementi di un vettore di interi i valori 0, 5, 10, 15, 20.

Analisi

Questo è un esercizio per capire come utilizzare i vettori: è possibile ovviamente scrivere 5 istruzioni di assegnazione per ciascun elemento del vettore, ma cerchiamo al solito di trovare una soluzione più generale.

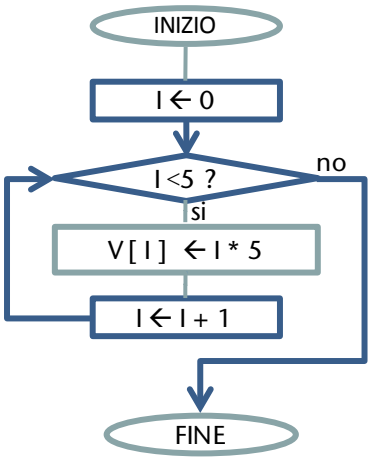
Posizione (indice)	valore
0	0
1	5
2	10
3	15
4	20

In particolare osserviamo la corrispondenza fra i valori da assegnare e le posizioni degli elementi ai quali vogliamo assegnare i valori stessi: possiamo ottenere il valore semplicemente moltiplicando la posizione per 5.

Tabella di descrizione delle variabili

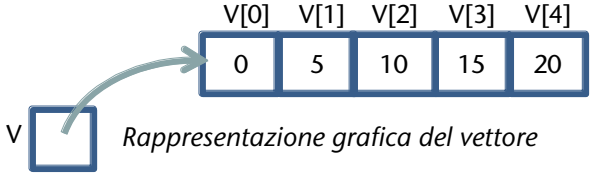
Nome	Descrizione	Tipo di dato
V[5]	Vettore	intero
i	Indice del vettore, cioè contatore che indica la posizione dell'elemento del vettore che si elabora	intero

Algoritmo



Codifica

```
using System;
class Program
    {
        static void Main(string[] args)
        {
            int[] V = new int[5];
            int i;
            for (i = 0; i < 5; i++)
            {
                V[i] = i * 5;
            }
        }
    }
```



Rappresentazione grafica del vettore

Tabella di traccia

I	V[0]	V[1]	V[2]	V[3]	V[4]
0					
	0				
1					
		5			
2					
			10		
3					
				15	
4					
					20
5					

4.3.1.6. Operazioni con i vettori

Cerchiamo di comprendere più a fondo i vettori eseguendo alcune operazioni su di essi: per quanto sappiamo fin qui è necessario intervenire sui singoli elementi, in quanto non abbiamo ancora visto come lavorare con il vettore in quanto tale[58].

Considera un vettore di 5 interi ed effettuane una copia.

Tabella di descrizione delle variabili

Nome	Descrizione	Tipo di dato
A[5]	vettore A da copiare	intero
B[5]	vettore B su cui effettuare la copia	intero
I	posizione degli elementi	

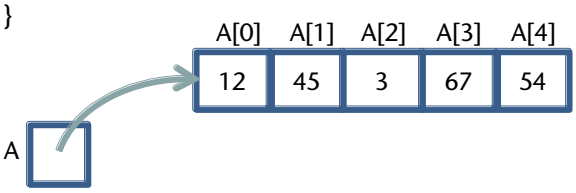
esempio di valori da copiare

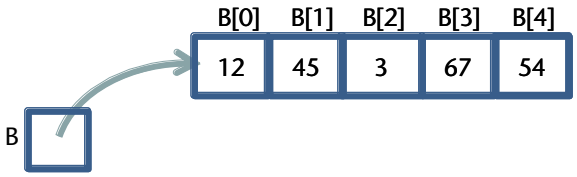
12	A[0]	0	B[0]	12
45	A[1]	1	B[1]	45
3	A[2]	2	B[2]	3
67	A[3]	3	B[3]	67
54	A[4]	4	B[4]	54

osserva: gli elementi corrispondenti dei due vettori hanno la stessa posizione

La tecnica corretta per risolvere questo problema è quella di esaminare ad uno ad uno gli elementi del primo vettore e copiare i valori sul secondo vettore.

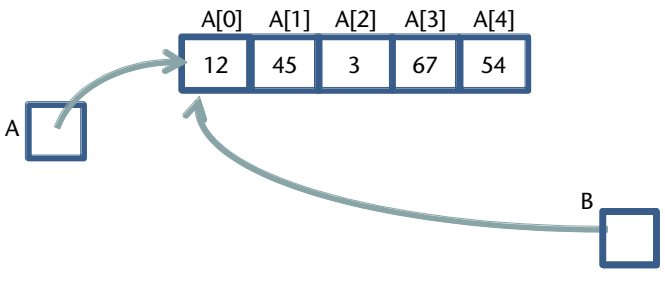
```
using System;
class Vettori
{   static void Main()
   {  int[] A = { 12, 45, 3, 67, 54 };
      int[] B = new int[5];
      int i;
      for (i = 0; i < 5; i++)
      {
         B [i] = A [i];
      }
   }
}
```





Considera il seguente esempio ed osserva cosa capita!

```
using System;
class Vettori
{   static void Main()
   {
      int[] A = { 12, 45, 3, 67, 54 };
      int[] B = new int[5];
      B = A;
   }
}
```

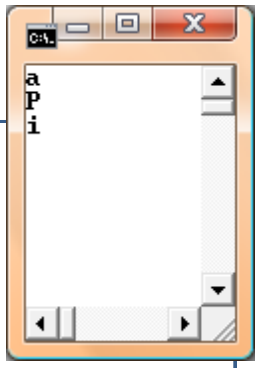


[58] Si tratterebbe di considerare il vettore come oggetto e fare riferimento a proprietà e metodi, secondo il paradigma della OOP.

4.3.1.7. Vettori e stringhe

Vi è una stretta relazione fra i vettori di caratteri e le stringhe, in quanto in memoria una stringa viene memorizzata proprio come sequenza di caratteri: di qui deriva che si può individuare ogni singolo carattere specificandone la posizione (vedi paragrafo 4.2.1.4).

Osserva e leggi il codice corrispondente




```
1  using System;
2  class CaratteriStringhe
3  {   static void Main()
4     {   string S = "Pippo";
5         char c = 'a';
6         Console.WriteLine(c);
7
8         c = S[0];   // un elemento di una stringa è un carattere
9         Console.WriteLine(c);
10
11        c = Convert.ToChar(S.Substring(1, 1));
12        //una stringa formata da un solo carattere NON è un tipo carattere
13        // è necessaria la conversione
14        Console.WriteLine(c);
15
16        // c = (char) S.Substring(1,1); ERRATO
17        // Impossibile convertire il tipo 'string' in 'char'.
18
19        Console.ReadLine();
20    }
21 }
```

L' esempio precedente indica che stringa e vettore di caratteri sono comunque due tipi diversi.

Consideriamo ora un' altra differenza osservando il comportamento degli *operatori di uguaglianza e di disuguaglianza*: nel caso dei vettori essi si riferiscono al puntatore, come si è visto nel paragrafo precedente, *nel caso delle stringhe invece operano direttamente sui valori*, rendendo così più immediato il loro utilizzo.

```
1  class Program
2  {
3     static void Main()
4     {
5        char[] va = { 's', 'a', 'l', 'v', 'e' };
6        char[] vb = { 's', 'a', 'l', 'v', 'e' };
7        char[] vc = { 'c', 'i', 'a', 'o'};
8
9        Console.WriteLine(va == vb); //no
10       Console.WriteLine(va == vc); //no
11
12       vc = va; //vc prende il valore di va, cioè si sposta il riferimento
13       Console.WriteLine(vc);  //visualizza salve
14       vc[0] = 'h';      //sa modifica il suo valore
15       vc[1] = 'e';
16       vc[2] = 'l';
17       vc[3] = 'l';
18       vc[4] = 'o';
19       Console.WriteLine(va);  //visualizza hello
20
21       string sa = "salve";
22       string sb = "salve";
23       string sc = "ciao";
25
26       Console.WriteLine(sa == sb); //si
27       Console.WriteLine(sa == sc); //no
28
29       sc = sa;  // sc prende il valore di sa, cioè salve
30       Console.WriteLine(sc); //visualizza salve
31       sc = "hello";          // sa mantiene il valore precedente
32       Console.WriteLine(sa);  //visualizza salve
33       Console.ReadLine();
34    }
35 }
```

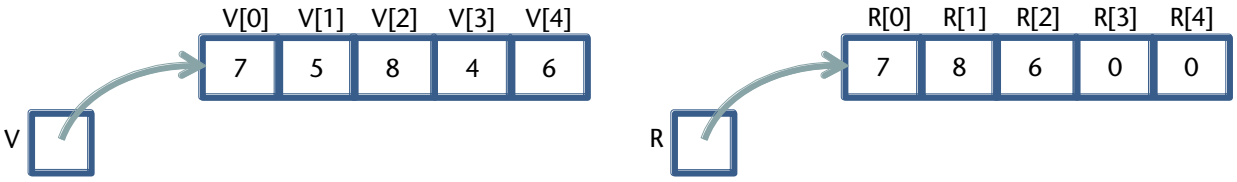
4.3.2. Utilizzare i vettori

Vengono mostrati alcuni esercizi con i vettori, quindi un esempio di applicazione per la soluzione di un semplice problema con l'uso di array per la gestione dei dati.

4.3.2.1. Selezionare gli elementi di un vettore

Assegnare ai 5 elementi di un vettore di voti i valori 7,5,8,4,6 e copiare su un altro vettore solo i voti positivi memorizzandoli uno di seguito all' altro.

Analisi
Rappresentiamo graficamente la situazione iniziale ed il risultato che vogliamo ottenere.



Per risolvere il problema dovremo esaminare ad uno ad uno tutti i valori del primo vettore quindi, solo per quelli che risultano positivi, provvedere a copiarli nel secondo vettore, a partire dalla prima posizione, spostandoci sempre nella posizione successiva. Vediamo quindi che ad esempio il valore 7 si trova nella stessa posizione dei due vettori, ma non altrettanto si può dire del valore 8.

Tabella di descrizione delle variabili

Nome	Descrizione	Tipo di dati
V[5]	Vettore con i voti iniziali	intero
R[5]	vettore con i risultati, cioè con i voti positivi	intero
i	Indice degli elementi del vettore con i voti iniziali	intero
j	Indice degli elementi del vettore che memorizza i risultati, cioè i soli voti positivi	intero

Codifica

```
using System;
class Program
{
    static void Main( )
    {
        int [ ] V = {7,5,8,4,6};
        int [ ] R = new int [5];
        int i, j=0;
        for (i = 0; i < 5; i++)
        {
            if (V[i] >= 6)
            {
                R[j] = V[i];
                j = j + 1;
            }
        }
    }
}
```

4.3.2.2. Gestione alunni (vettori paralleli e presentazione menu)

Problema→ Oggi vengono interrogati 5 alunni, a ciascuno di essi viene assegnato un voto (numero intero fra 1 e 10): interessa poter inserire prima l'elenco dei nomi, di seguito l'elenco dei voti, quindi visualizzare un elenco completo di nomi e voti ed inoltre conoscere sia il numero di coloro che hanno valutazioni positive sia l'elenco dei nomi di coloro che devono invece recuperare.

Analisi (prima versione)
Il problema è descritto in dettaglio, non ci sono ambiguità e sono specificate sia le richieste, cioè il risultati che si vogliono ottenere, sia i dati che si hanno a disposizione.
Si propone una prima soluzione molto semplice, in cui l'utente viene guidato dal programma nell'inserimento dei dati, quindi vengono mostrati gli output richiesti.

Tabella di descrizione delle variabili

Nome	Descrizione	tipo	Input/Output
Nomi[5]	Vettore dei nominativi	stringa	Input/Output
V[5]	Vettore dei voti	intero	Input/Output
B	Contatore numero studenti Bravi, con voto almeno sufficiente	intero	output
i	Contatore per indicare la posizione dell'elemento che si esamina	intero	lavoro

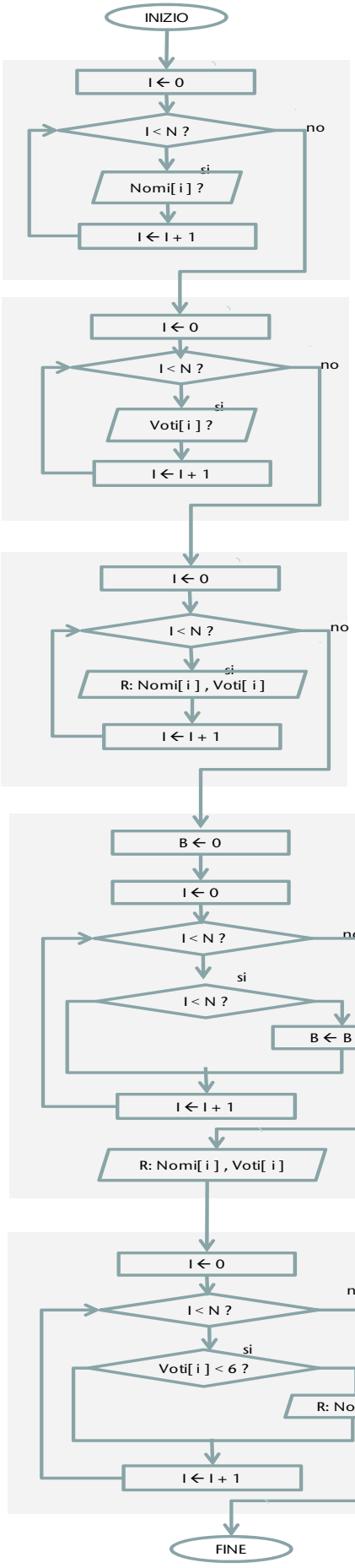
Le tabelle seguenti mostrano un esempio di possibili valori: essendo dati di tipo differente, stringhe per i nomi e interi per i voti, è necessario utilizzare due diverse variabili[59].

Osserviamo comunque che vi è una corrispondenza fra gli elementi che nei due elenchi si trovano nella stessa posizione: potremo quindi utilizzare una stessa variabile per indicare la posizione del nome e del voto relativi ad un certo alunno, anche se i valori sono memorizzati in due differenti vettori[60].

Nomi[0]	giuseppe
Nomi[1]	elisa
Nomi[2]	maria
Nomi[3]	matteo
Nomi[4]	aldo

Voti [0]	4
Voti [1]	7
Voti [2]	8
Voti [3]	8
Voti [4]	5

[59] Tale affermazioni è vera se riferita alle nostra ttuali conoscenze tecniche, pi avanti si vedrano altre soluzioni.
[60] Ci si riferisce talvolta a questa sistuazione con l' espressione vettori paralleli.



Algoritmo (prima versione)

Di seguito viene presentato l'algoritmo, mettendone in evidenza i blocchi corrispondenti ad ogni specifica operazione: ciascuno di tali blocchi viene indicato con un identificatore che lo contraddistingue.

inserimento nomi

inserimento voti

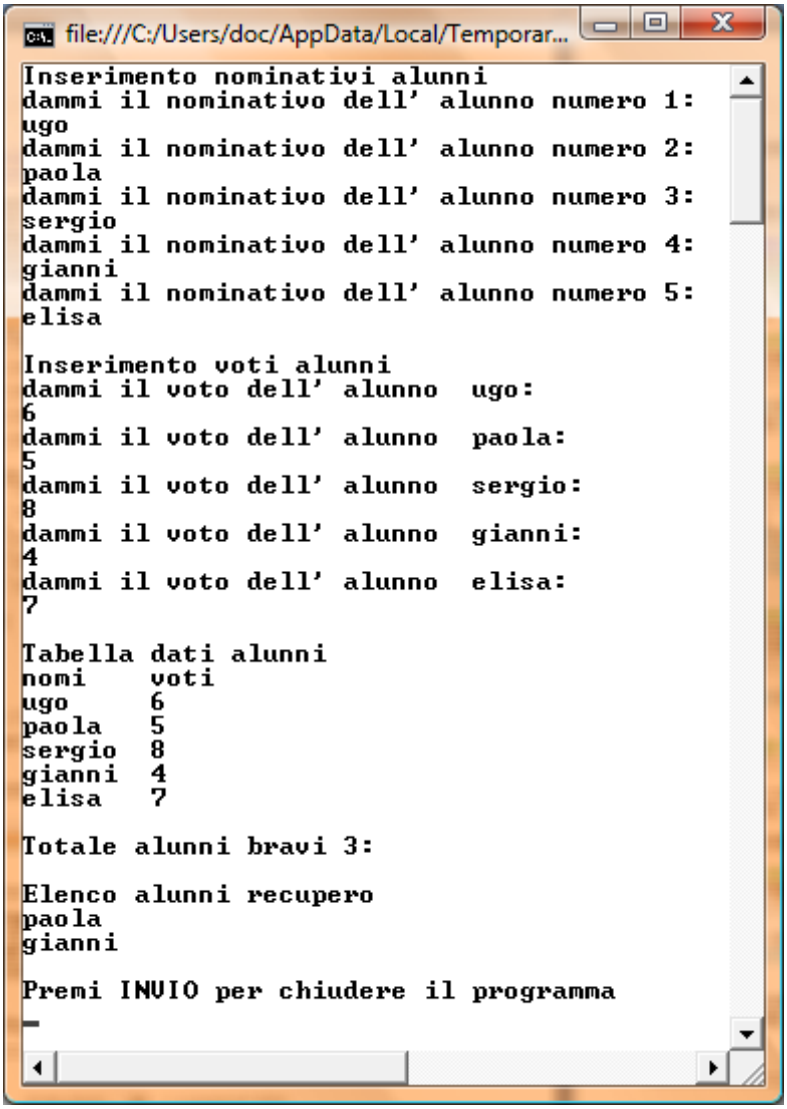
Visualizzazione dati

Conteggio alunni bravi

Elenco alunni recupero

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int[] V = new int[5];
        string[] Nomi = new string[5];
        int i;            //contatore
        int B = 0; // contatore numero bravi
        string tmp;

        //inserimento nomi
        Console.WriteLine("Inserimento nominativi alunni");
        for (i = 0; i < 5; i++)
        {
            Console.WriteLine("dammi il nominativo dell' alunno numero {0}: ", i + 1);
            Nomi[i] = Console.ReadLine();
        }

        //inserimento voti
        Console.WriteLine("\nInserimento voti alunni");
        for (i = 0; i < 5; i++)
        {
            Console.WriteLine("dammi il voto dell' alunno  {0}: ", Nomi[i]);
            tmp = Console.ReadLine();
            V[i] = Convert.ToInt32(tmp);
        }

        //visualizzazione dati
        Console.WriteLine("\nTabella dati alunni");
        Console.WriteLine("nomi\tvoti");
        for (i = 0; i < 5; i++)
        {
            Console.WriteLine(Nomi[i] + "\t" + V[i]);
        }

        //conteggio alunni bravi
        for (i = 0; i < 5; i++)
        {
            if (V[i] >= 6)
                B = B + 1;
        }
        Console.WriteLine("\nTotale alunni bravi {0}: ", B);

        //elenco alunni recupero
        Console.WriteLine("\nElenco alunni recupero");
        for (i = 0; i < 5; i++)
        {
            if (V[i] < 6)
                Console.WriteLine(Nomi[i]);
        }
        Console.WriteLine("\nPremi INVIO per chiudere il programma");

        Console.ReadLine();
    } //fine Main
}//fine classe
```

Output a video



Analisi (seconda versione)

Il problema è risolto, però è bene migliorare **l'interfaccia** della nostra **applicazione**, cioè il modo con cui essa si presenta all'utente. Naturalmente non modifichiamo invece la logica della applicazione, che risulta corretta, in quanto ha risolto in modo efficace il problema.

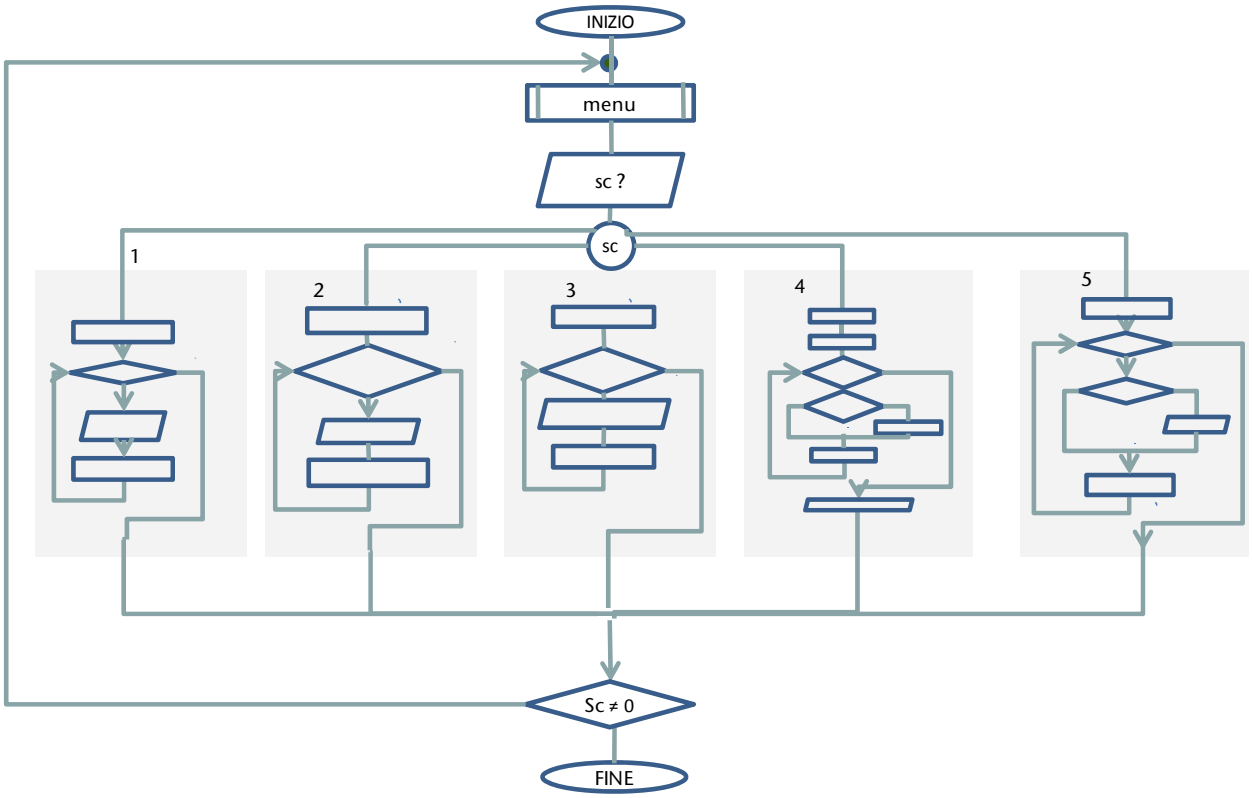
Mostriamo quindi una seconda versione della nostra applicazione, che all'avvio presenta un **menu** dal quale l'utente possa attivare le diverse funzionalità: inserimento dei dati, visualizzazione tabella con nomi e voti, conteggio alunni bravi e selezione nominativi da avviare al recupero.

La gestione del menu è un problema classico, la cui soluzione è già stata mostrata (vedi paragrafo 3.3.3.2).

La tabella delle variabili è pressoché invariata rispetto alla soluzione precedente, viene solo introdotta una nuova variabile sc che permette di memorizzare la scelta effettuata dall'utente fra le diverse opzioni del menu.

Osserviamo che a ciascuna voce del menu corrisponde un preciso piccolo problema, che si configura come sottoproblema del problema principale: ciascuno di questi sottoproblemi verrà gestito in modo autonomo , tenendo conto comunque conto del fatto che ognuno di essi opera su dati comuni al problema generale.

Algoritmo (seconda versione)



Lo stesso algoritmo, rappresentato in forma ancora più sintetica

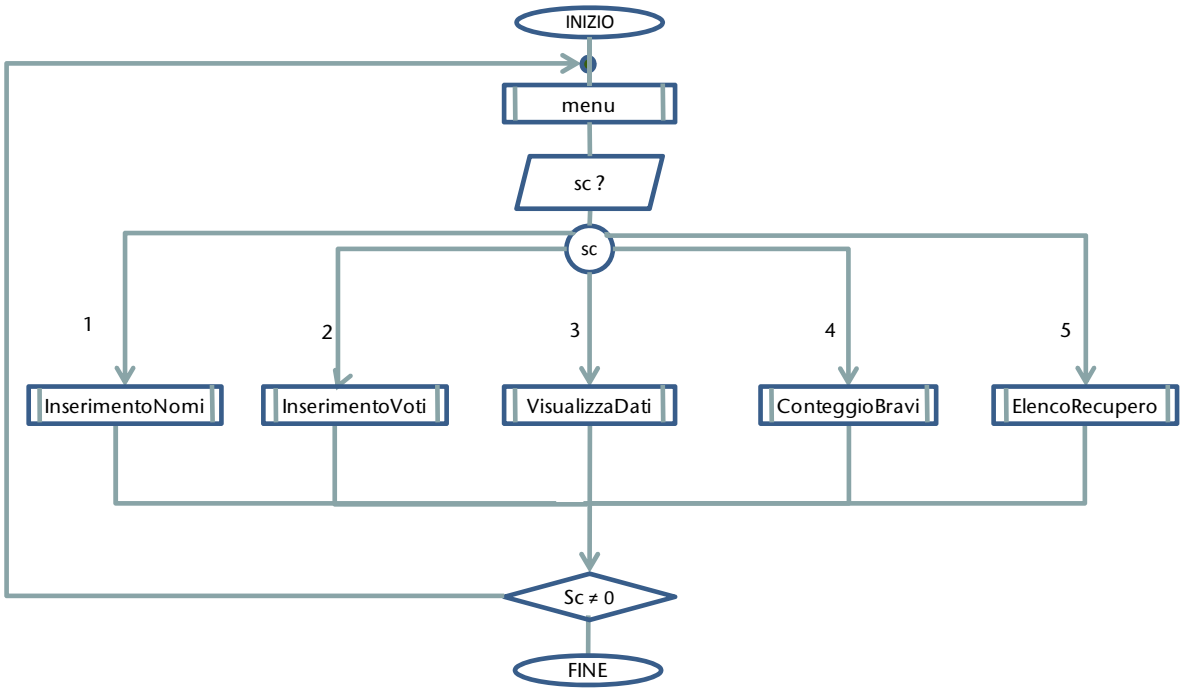


Figura 4-5 - Algoritmo gestione elenco nominativi e voti

Il blocco ripetizione consente di presentare il menu e permettere all'utente di scegliere fra le diverse opzioni fino a quando è interessato all'uso del programma.

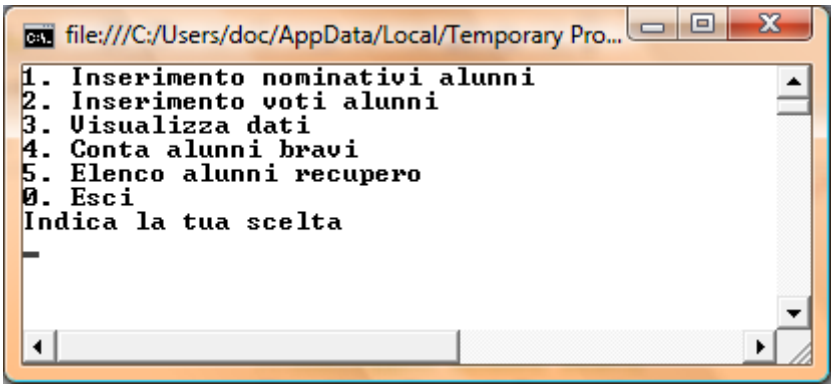
I diversi bloccchi corrispondenti alla logica del problema non sono modificati, e sono inseriti ciascuno in corrispondenza della rispettiva scelta: nel secondo diagramma sono indicati in forma più sintetica, precisandone il nome che li identifica (vedi la discussione sul metodo top-down **5.1**).
Il blocco menu ha lo scopo di presentare all'utente le diverse opzioni.

Codifica

```
using System;
class Program
{
   static void Main( )
   {
      int[] V = new int[5];
      string[] Nomi = new string[5];
      int i;            //contatore
      int B = 0; // contatore numero bravi
      int sc;    //scelta opzione menu
      string tmp;
      do
      {
         // presentazione menu
         Console.WriteLine("1. Inserimento nominativi alunni");
         Console.WriteLine("2. Inserimento voti alunni");
         Console.WriteLine("3. Visualizza dati");
         Console.WriteLine("4. Conta alunni bravi");
         Console.WriteLine("5. Elenco alunni recupero");
         Console.WriteLine("0. Esci");
        Console.WriteLine("Indica la tua scelta");
         tmp = Console.ReadLine();
         sc = Convert.ToInt32(tmp);
         switch( sc)
         {
            case 1:
              //inserimento nomi
              Console.WriteLine("Inserimento nominativi alunni");
              for (i = 0; i < 5; i++)
              {
              Console.WriteLine("dammi il nominativo dell' alunno numero {0}: ", i + 1);
              Nomi[i] = Console.ReadLine();
              }
               break;

            case 2:
              //inserimento voti
              Console.WriteLine("\nInserimento voti alunni");
              for (i = 0; i < 5; i++)
              {
                  Console.WriteLine("dammi il voto dell' alunno  {0}: ", Nomi[i]);
                  tmp = Console.ReadLine();
                  V[i] = Convert.ToInt32(tmp);
              }
               break;
```

```
                case 3:
                  //visualizzazione dati
                  Console.WriteLine("\nTabella dati alunni");
                  Console.WriteLine("nomi\tvoti");
                  for (i = 0; i < 5; i++)
                  {
                     Console.WriteLine(Nomi[i] + "\t" + V[i]);
                  }
                   break;

                case 4:
                  //conteggio alunni bravi
                  for (i = 0; i < 5; i++)
                  {
                      if (V[i] >= 6)
                        B = B + 1;
                  }
                  Console.WriteLine("\nTotale alunni bravi {0}: ", B);
                   break;

                case 5:
                  //elenco alunni recupero
                  Console.WriteLine("\nElenco alunni recupero");
                  for (i = 0; i < 5; i++)
                  {
                     if (V[i] < 6)
                        Console.WriteLine(Nomi[i]);
                  }
                   break;
            } //fine switch
        } while (sc != 0);

        Console.WriteLine("\nPremi INVIO per chiudere il programma");

        Console.ReadLine();
    }
}
```

Output a video



Versione con record, vedi paragrafo 4.5.1.3

Versione con metodi, vedi paragrafo 5.2.2.1

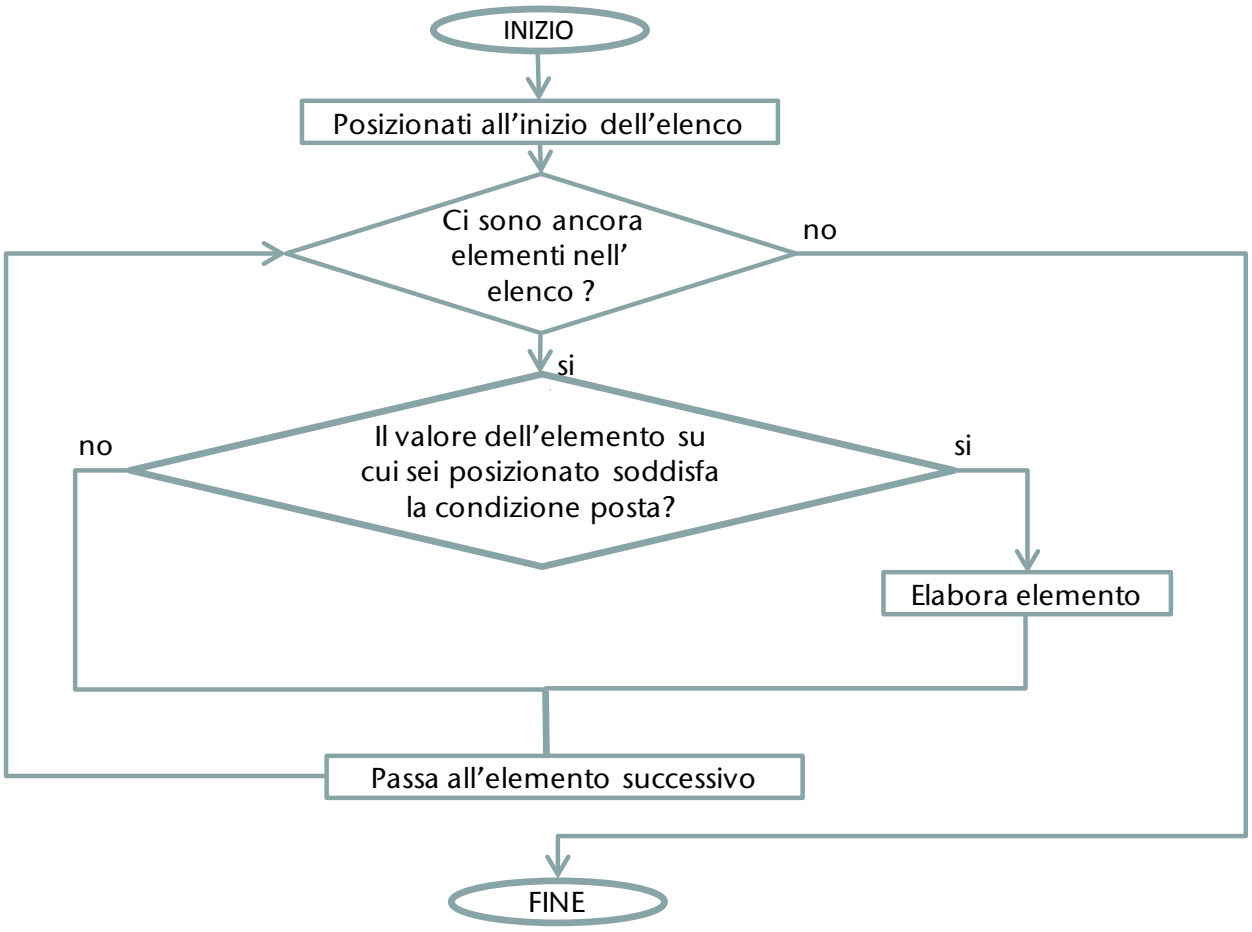
4.3.3. Elaborazioni classiche con i vettori

Si propongono qui alcuni problemi di base fornendo soluzioni classiche che, seppur semplici, si ritengono significative per la formazione di base di uno studente che si occupa di informatica.

4.3.3.1. Selezione

Problema→ selezionare tutti gli elementi di un elenco che soddisfano una certa condizione. Ad esempio in un insieme di voti selezionare quelli sufficienti.

Il problema è formulato in termini generali, in particolare non è specificato cosa si intende fare degli elementi selezionati: potremmo voler contare quanti sono, oppure visualizzarli, o quant'altro. Si propone un metodo di soluzione generale, che può essere utilizzato con qualsiasi tipo di dato e con qualsiasi numero di valori.



Mostriamo l'implementazione di un caso particolare, scegliendo di visualizzare i valori che soddisfano la condizione posta. Dobbiamo inoltre pensare a come memorizzare i dati: ci si riferisce come esempio ad un vettore costituito da 10 elementi di tipo intero.

Leggi il codice

```
using System;
class Program
{  static void Main( )
   {   int[] v = { 6,7,5,8,4,6,6,5,8,4,};
       for (int i = 0; i < 10; i++)
       {
          if (v[i] >=6)
          {
             Console.WriteLine("voto sufficiente: {0}", v[i]);
          }
       }
   }
}
```

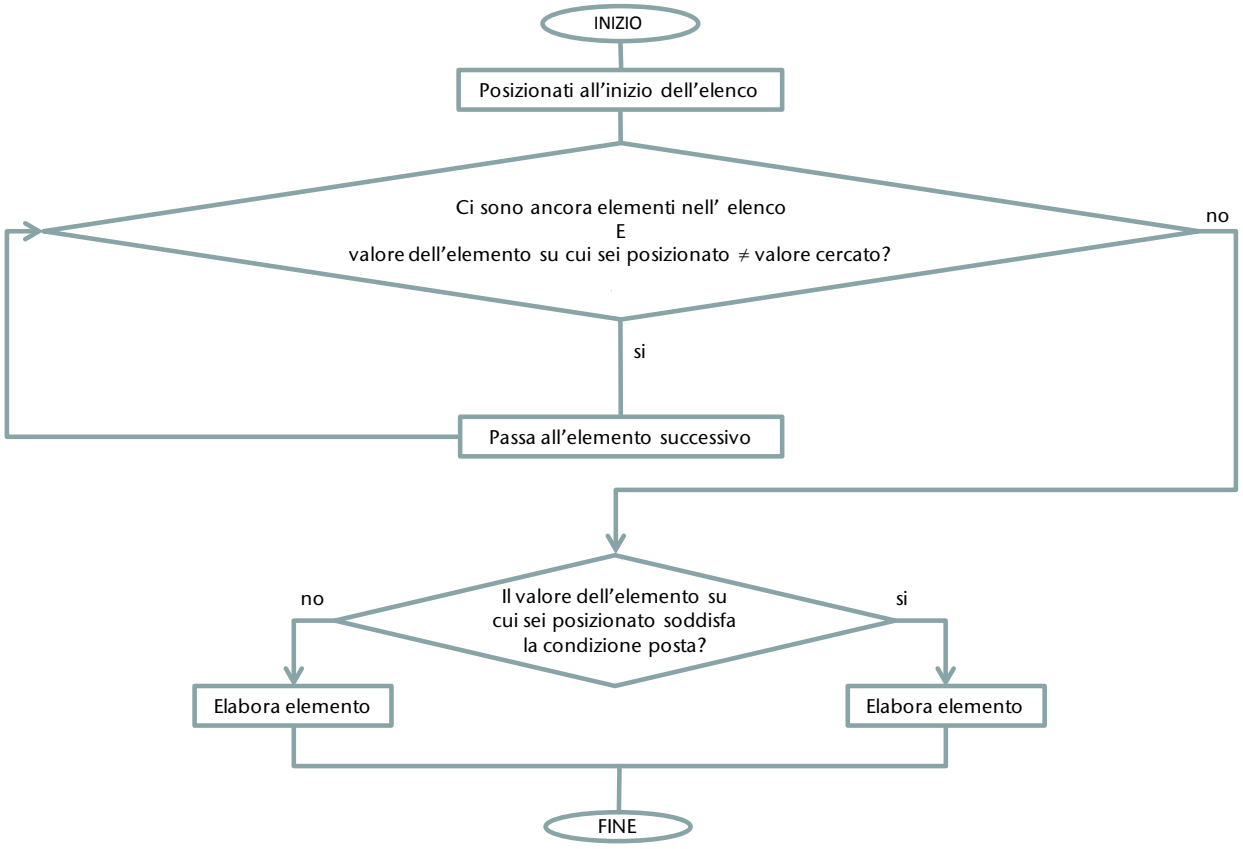
4.3.3.2. Ricerca sequenziale

Problema → Verificare se un valore X è presente in un elenco. Ad esempio in un insieme di nomi cerca "Matteo".

Osserviamo la differenza fra **selezione** e **ricerca**: nel primo caso abbiamo individuato tutti gli elementi che soddisfano una condizione, ora si tratta di individuare, se è presente, un elemento, quello ricercato, in un insieme dove gli elementi sono tutti diversi fra loro.
Proponiamo una soluzione generale, considerando che dovremo esaminare ad uno ad uno gli elementi e che dovremo interrompere la ricerca quando si verifica uno dei due seguenti casi:

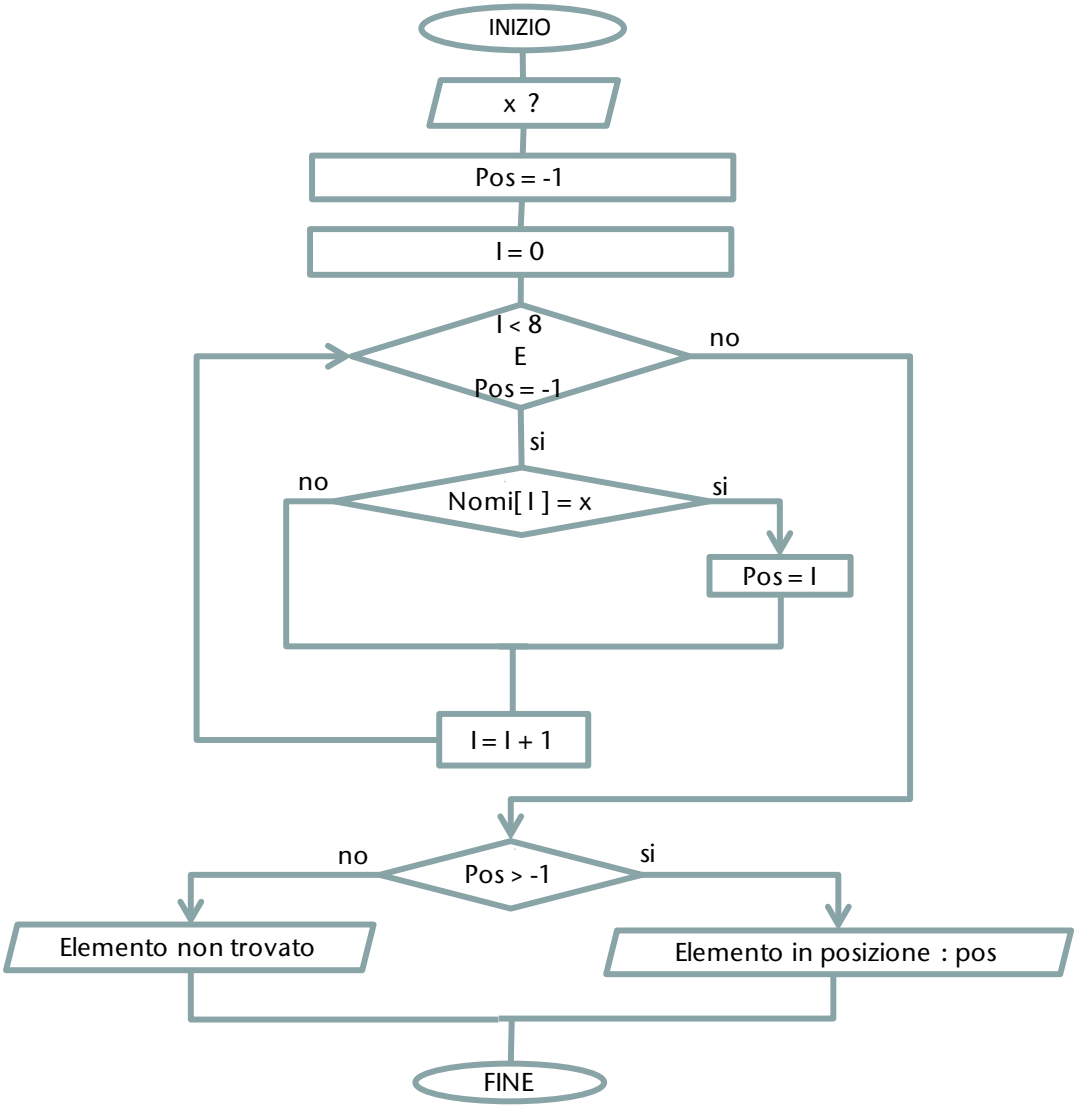
- l'elemento cercato è stato trovato
- tutti gli elementi dell'insieme sono stati controllati.

Naturalmente la ricerca può avere esito positivo, ma anche negativo, nel caso in cui l'elemento che sui sta cercando non sia presente nell'insieme dato.

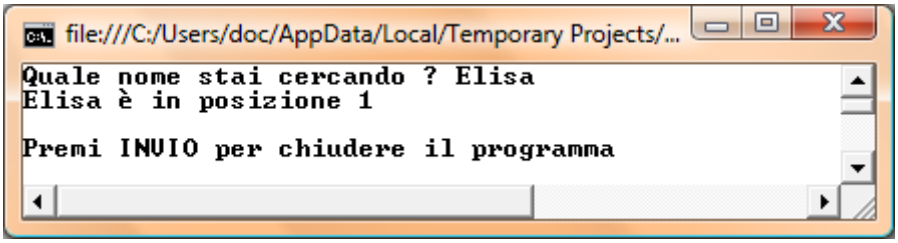


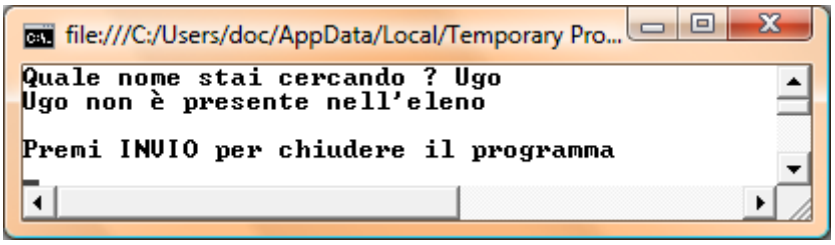
Questo algoritmo è molto generale, consideriamo ora un caso più specifico: **ricerchiamo l'elemento e visualizziamo la posizione in cui esso si trova**; qualora l'elemento non venga trovato si visualizza un messaggio informativo.

Supponiamo che i valori siano costituiti da un elenco di 8 nomi memorizzati su un vettore di stringhe, indichiamo con x il nome della variabile che memorizza il valore da cercare e con Pos la posizione dell'elemento trovato. La variabile Pos viene inizializzata a -1 ad indicare che, prima di iniziare la ricerca, l'elemento non è stato trovato!



Osserva e leggi il codice corrispondente






```
using System;
class Program
{
   static void Main(string[] args)
   {
      string[] Nomi = {"Giuseppe", "Elisa", "Piero", "Maria", "Gianni", "Elisabetta", "Matteo", "Manuela"};
      string x;
      Console.Write("Quale nome stai cercando ? ");
      x = Console.ReadLine();
      int Pos = -1;
      int I = 0;
      while ( (I<8) && (Pos == -1) )
      {
         if (Nomi[I] == x)
         {
            Pos = I; //ho trovato l'elemento in posizione I
         }
         I = I + 1;
      }
      if ( Pos > -1)
      {
         Console.WriteLine("{0} è in posizione {1}",x,Pos);
      }
      else
      {
         Console.WriteLine("{0} non è presente nell'eleno",x);
      }

      Console.WriteLine("\nPremi INVIO per chiudere il programma");
      Console.ReadLine();

   }
}
```

4.3.3.3. Ricerca binaria

Problema→ Cercare un valore X in un vettore ordinato.

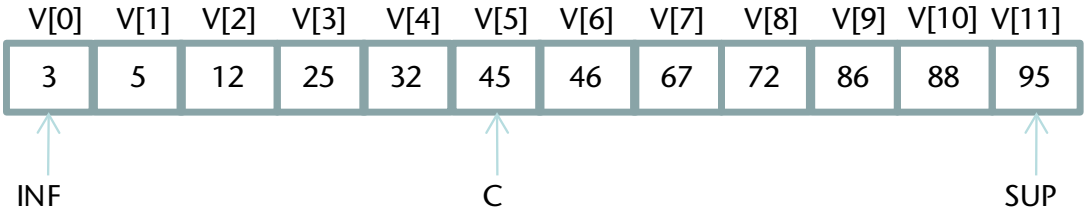
Consideriamo ad esempio un vettore con dati di tipo intero in ordine crescente, i valori potrebbero essere quelli rappresentati in figura, e proviamo a cercare in esso un dato qualsiasi, ad esempio 88.

Rispetto al problema tradizionale della ricerca qui c'è una variante interessante: sappiamo che i dati sono ordinati, quindi è ragionevole nella ricerca di una soluzione del problema avvalerci della nuova informazione al fine di rendere più efficiente l'algoritmo.

È chiaro che i dati proposti sono solo esemplificativi, dobbiamo trovare una soluzione generale, che possa essere valida qualsiasi siano i valori del vettore, e qualunque sia il valore da cercare.

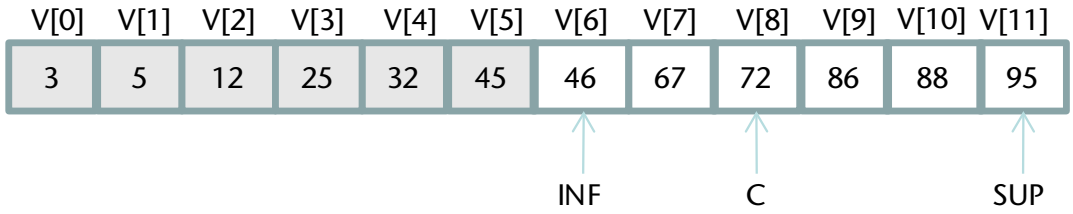
Quindi in generale noi non abbiamo alcuna indicazione relativa a dove si trova il valore, potrebbe essere in qualsiasi posizione.

Proviamo a vedere se per caso si trova nella posizione centrale (indicata con C), che calcolo come posizione intermedia fra l'estremo inferiore (INF=0) del vettore che sto considerando e l'estremo superiore (SUP = 11), scegliendo di arrotondare per difetto (questa scelta non è significativa, potrei arrotondare per eccesso); provo quindi a vedere se il valore cercato è nella posizione 5.

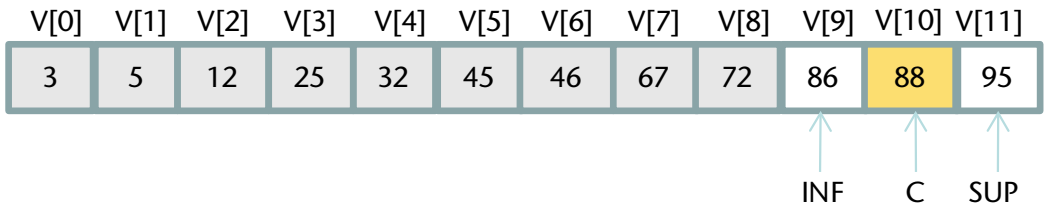


Osserva che in posizione 5 c'è il valore 45 che è minore del valore cercato: siccome i dati sono in ordine crescente, il valore 88, se è presente nel vettore, si trova sicuramente nella parte destra, dalla posizione 6 in avanti.

Nella figura qui di seguito viene rappresentata graficamente questa situazione: gli elementi nelle prime posizioni non sono più da prendere in considerazione (per questo sono stati oscurati), l'indicazione INF che rappresenta la posizione inferiore da cui cominciare a ricercare è stata spostata in corrispondenza dell'elemento in posizione 6. L'estremo superiore dell'intervallo di ricerca rimane invariato, per quanto ne sappiamo fin qui il valore 88 potrebbe benissimo trovasi proprio nell'ultima posizione del vettore. Come prima, iniziamo a cercare in mezzo, cioè individuiamo la posizione centrale fra gli elementi ancora da considerare (il valor medio fra 6 e 11, le due posizioni estreme di ricerca) che risulta essere la posizione 8.



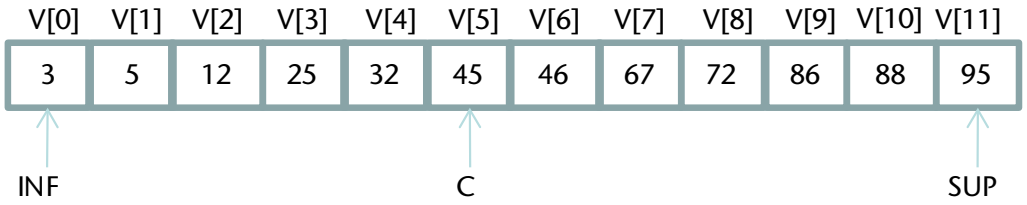
Come sopra, il procedimento è sempre lo stesso! Nella posizione 8 c'è il valore 72, quindi il valore 88, se c'è, si trova oltre la posizione 6. La figura rappresenta la nuova situazione.

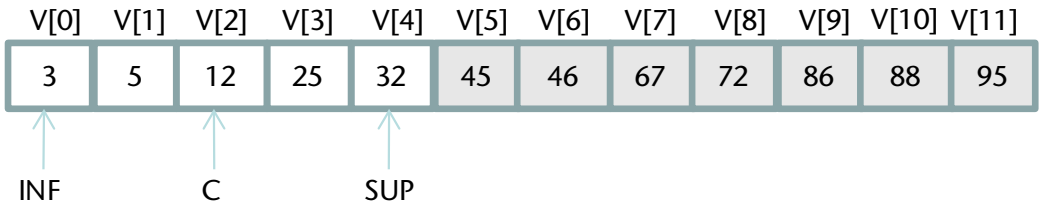


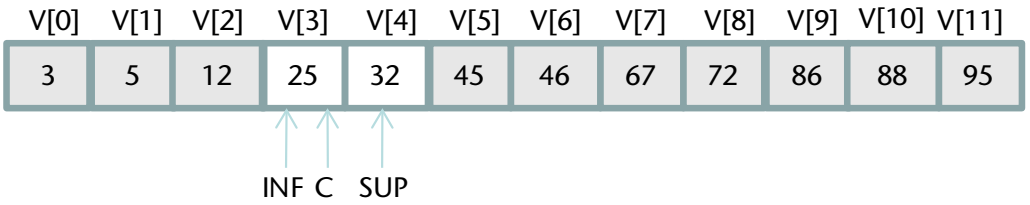
Aggiorniamo la posizione degli indicatori INF e SUP, quindi calcoliamo la nuova posizione centrale, che risulta essere la posizione 10.
Verifichiamo se troviamo l'elemento che stiamo cercando: questa volta l'esito è positivo, la ricerca si è conclusa con successo.

Da dove deriva il nome di ricerca binaria? Rifletti!
il vettore è costituito da 12 elementi, e fossimo stati molto fortunati avremmo potuto trovare il valore cercato al primo tentativo.
Da notare che però, pur avendo eseguito una sola prova, siamo sicuri che l'elemento, se c'è, si trova da una sola parte, quindi possiamo già scartare metà degli elementi! La lunghezza del vettore su cui dobbiamo ancora effettuare la ricerca si è dimezzata!
Questo avviene ad ogni tentativo: il secondo lo effettuo considerando solo più 6 elementi, non sono ancora fortunato, ma di nuovo metà degli elementi può essere scartata.
È chiaro che questo metodo è più efficiente della ricerca sequenziale (ha un costo però nemmeno tanto nascosto: i dati devono essere ordinati).

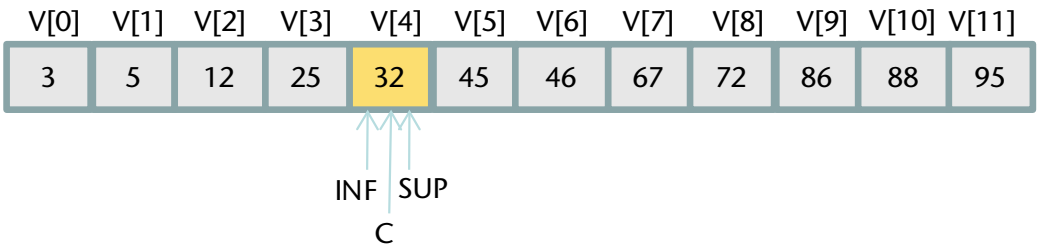
Effettuiamo un'altra prova ricercando il valore 32: si riporta solo la simulazione grafica, il ragionamento è quello precedente.





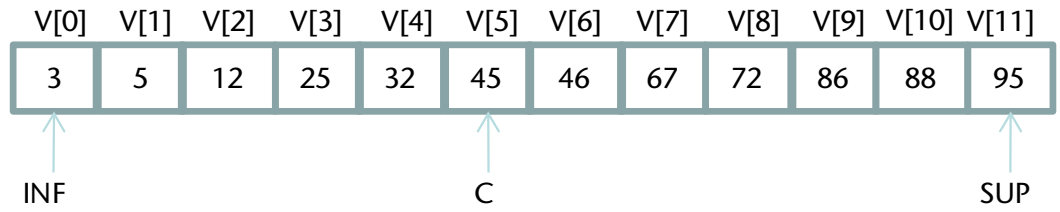


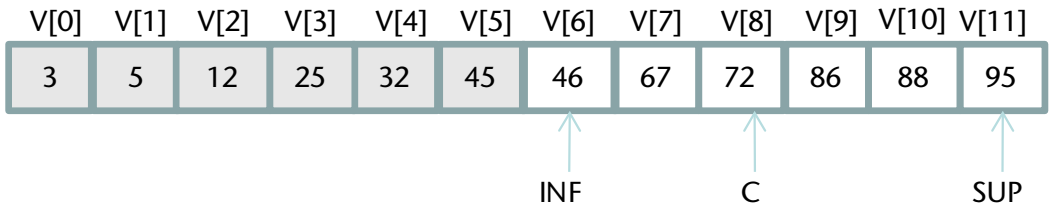
Ancora non abbiamo trovato il nostro elemento, andiamo avanti con il solito metodo: spostiamo l'estremo inferiore a sinistra (il valore se c'è è maggiore di 25), e calcoliamo la nuova posizione del massimo.

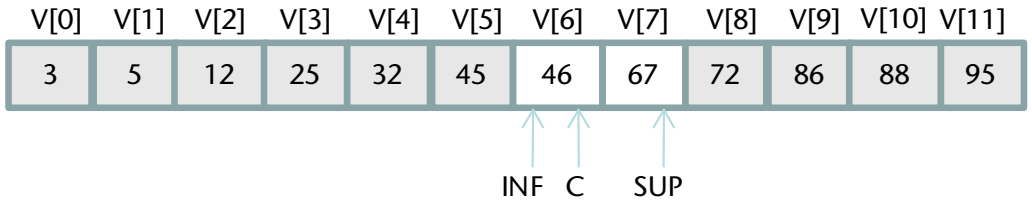


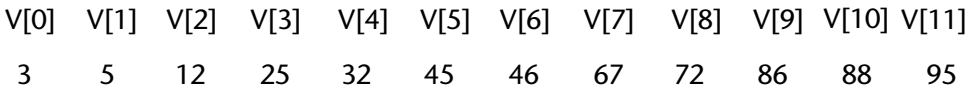
Anche questa volta la ricerca si conclude con un successo.

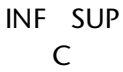
Cosa succede quando cerchiamo un valore che non è presente? Non possiamo sempre essere fortunati! Proviamo a seguire il metodo della ricerca binaria per vedere se è presente il valore 70.





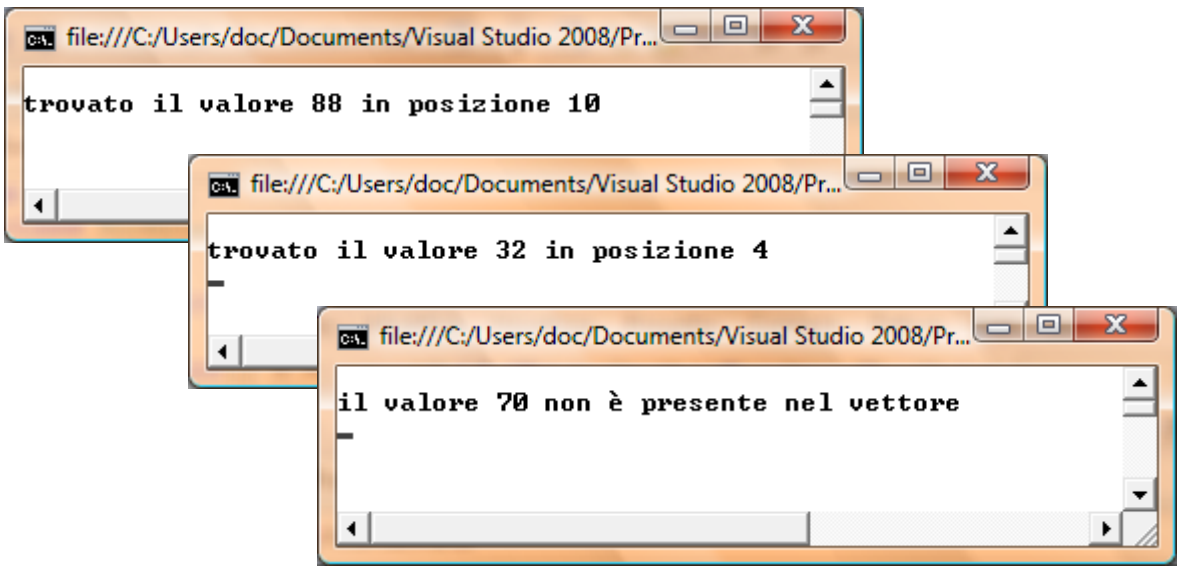






Ancora una volta la posizione dell'estremo inferiore e superiore di ricerca coincidono, cioè la ricerca è ormai limitata ad un solo elemento. Neanche in questa posizione abbiamo trovato il valore 70 che stiamo cercando, allora possiamo affermare con sicurezza che esso non è presente nel vettore.

Osserva e leggi il codice corrispondente



```
1   class RicercaBinaria
2   {
3      static void Main()
4      {
5         int[] v = { 3, 5, 12, 25, 32, 45, 46, 67, 72, 86, 88, 95 };
6         int INF = 0, SUP = 11, C;
7         int X= 88;
8         C = (INF + SUP) / 2;
9         while ((v[C] != X) && (INF < SUP))
10        {
11           if (v[C] > X)
12           {
13              SUP = C - 1;
14           }
15           else
16           {
17              INF = C + 1;
18           }
19           C =  (INF + SUP) / 2;
20        }
21        if (v[C] == X)
22        {
23           Console.WriteLine("\ntrovato il valore {0} in posizione {1}", X, C);
24        }
25        else
26        {
27           Console.WriteLine("\nil valore {0} non è presente nel vettore", X);
28        }
29        Console.ReadLine();
30     }
31  }
```

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 9	la ricerca del valore prosegue finché non lo abbiamo ancora trovato (v[C] != X) purché ci siano ancor posizioni del vettore da esaminare (INF < SUP);
riga 21	all'uscita dal blocco ripetizione non possiamo sapere se abbiamo trovato l'elemento, o se esso non è proprio presente, quindi dobbiamo subito effettuare questo controllo.

Rifletti

Se raddoppia il numero di elementi del vettore, di quanto aumenta il numero di tentativi da effettuare? [Aumenta di uno, perché?]

Se ho ad esempio 1000 elementi, qual è il numero minimo di tentativi da effettuare? Quale il numero massimo? [Minimo 1, massimo 10: perché?]

4.3.3.4. Ordinamento

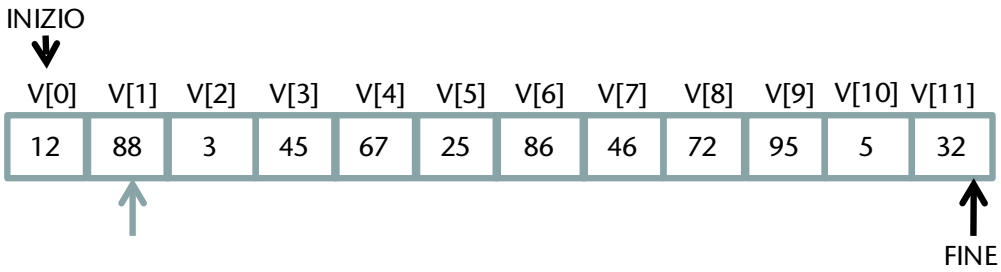
Problema→ Disporre in ordine crescente i valori di un vettore.

Consideriamo ad esempio il vettore di interi del paragrafo precedente in cui però i dati non sono disposti in ordine. Nel descrivere il procedimento risolutivo si di riferisce ad elementi del vettore in posizioni particolari, che descriviamo nella tabella seguente.

Tabella di descrizione delle variabili

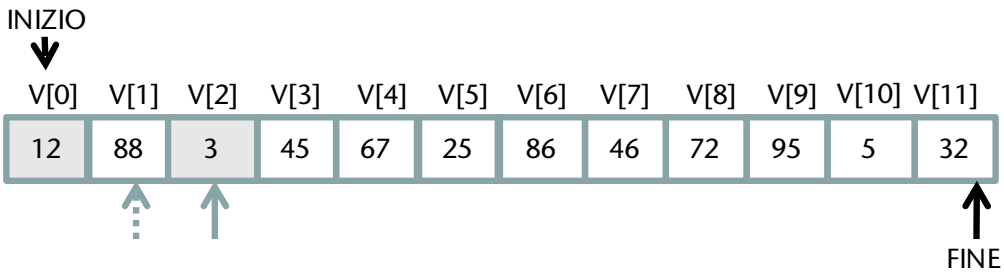
Nome	Descrizione
V[12]	vettore da ordinare
INIZIO	posizione del primo elemento del vettore da ordinare
FINE	posizione dell'ultimo elemento del vettore da ordinare
I	posizione dell'elemento corrente del vettore, cioè dell' elemento che via via stiamo elaborando.

Per risolvere il problema possiamo operare nel modo descritto di seguito: confrontiamo man mano ogni elemento del vettore con l'elemento nella posizione iniziale e, se necessario, effettuiamo uno scambio.

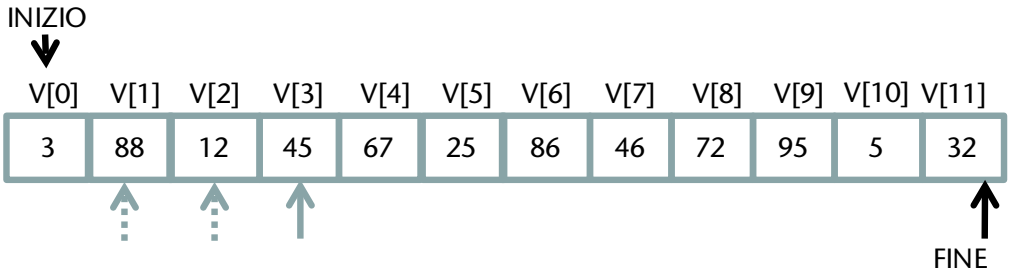


Osservo che l'elemento in posizione iniziale è V[0] e vale 12: confronto ogni elemento successivo con tale valore. Il primo confronto sarà fra V[0] e V[1]: il 12 è minore i 88, quindi i valori risultano essere in ordine crescente fa loro.

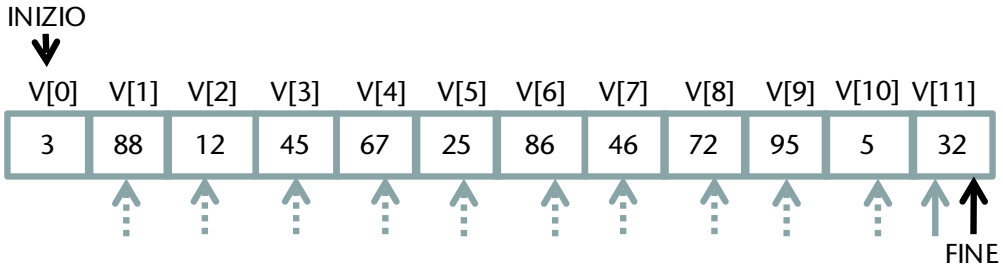
Passo all'elemento successivo, graficamente indicato dalla freccia che si sposta:



Ora i valori devono essere scambiati! Se voglio ottenere il vettore con i dati in ordine crescente i valori più piccoli si devono spostare all'inizio: effettuo lo scambio fra gli elementi evidenziati e passo a considerare l'elemento in posizione successiva.

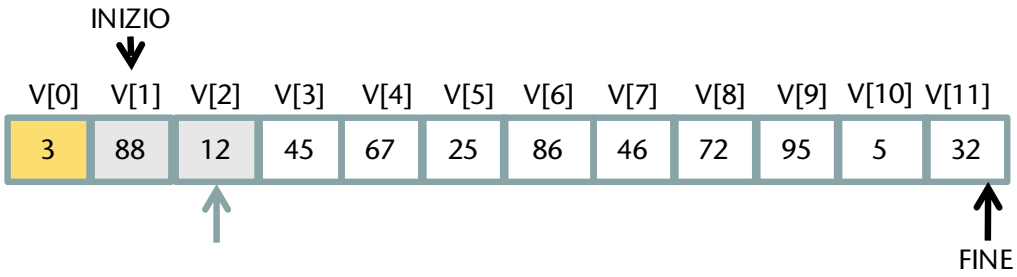


Confronto man mano tutti gli elementi con quello iniziale, se necessario effettuo gli scambi, fino ad arrivare all'ultimo elemento come indicato nella figura seguente.

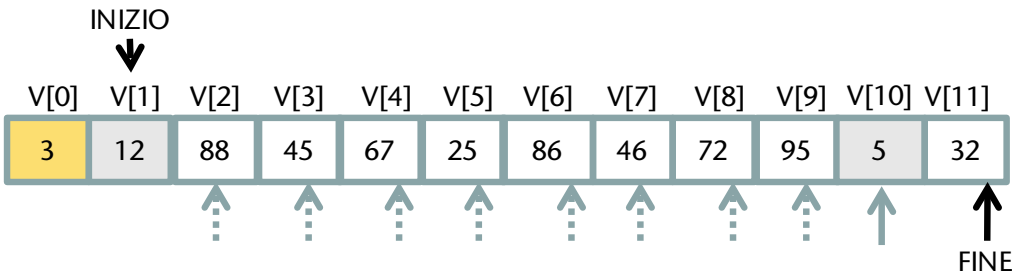


Riflettiamo su quanto è successo: ho confrontato tutti gli elementi con il primo, quando ho trovato un elemento minore di quello nella posizione iniziale ho effettuato lo scambio, quindi nella posizione iniziale si trova sicuramente il valore più piccolo. Il vettore non è ancora tutto ordinato, ma una parte sì: sono sulla buona strada, anche se il compito è piuttosto impegnativo.

Riprendo considerando non più tutti gli elementi del vettore, ma solo quelli ancora da mettere in ordine. Graficamente mettiamo in evidenza la parte del vettore già ordinata (solo il primo elemento per ora), e spostiamo la freccia inizio che indica la posizione dell'elemento iniziale del vettore (ancora) da ordinare.

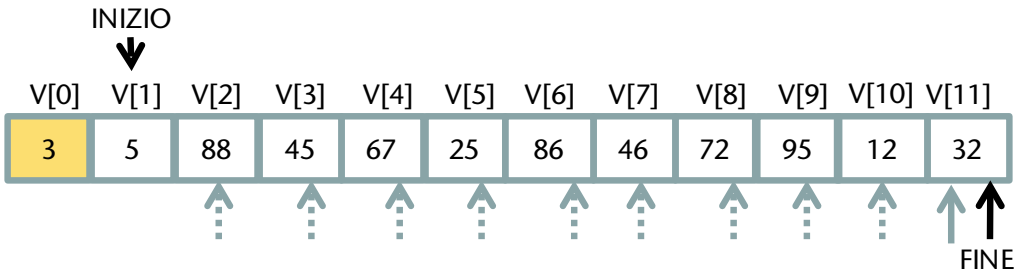


Come prima, confronto tutti gli elementi del vettore da ordinare, ad uno ad uno , con quello iniziale (che ora è quello in posizione 1): il valore 12 è minore di 88, quindi effettuo subito uno scambio! Poi vado avanti man mano.

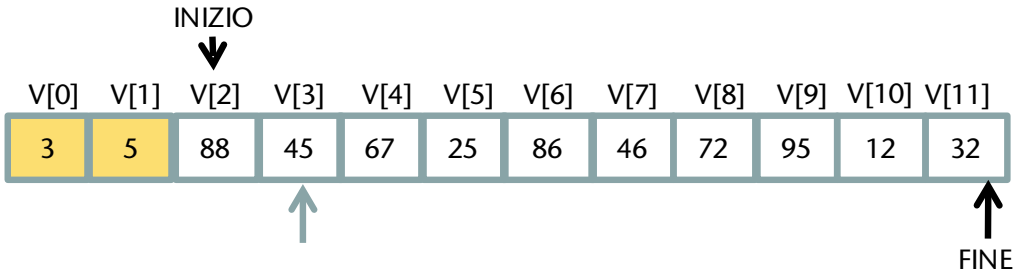


o.

Ho trovato un altro valore da scambiare: il 5, essendo più piccolo del 12, passa all'inizio, quindi si continua la scansione degli elementi che restano fino ad arrivare alla fine.[61]



Abbiamo terminato la seconda scansione, il risultato ottenuto è rappresentato in figura:



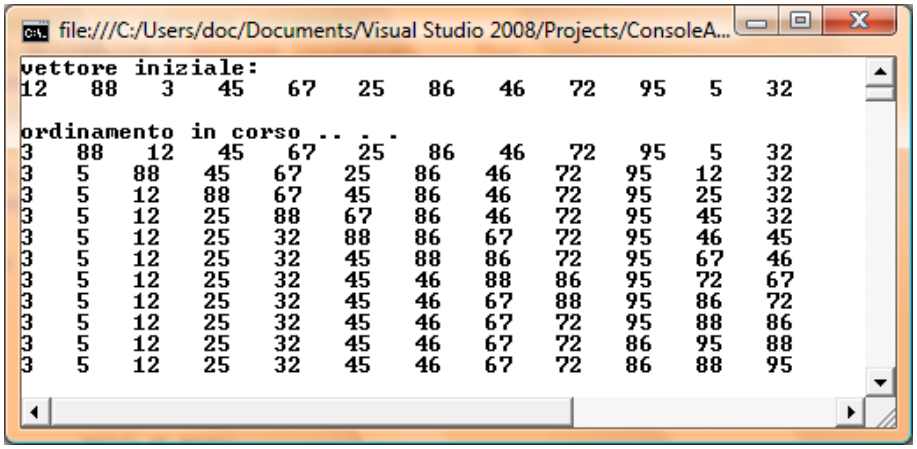
È chiaro che ogni volta che si fa una scansione completa si ottiene che il più piccolo degli elementi passa nella posizione iniziale, così alla scansione successiva si riduce il numero di elementi del vettore ancora da ordinare.

L'algoritmo risolutivo avrà quindi un blocco ripetizione corrispondente ad ogni singola scansione nella quale si elaborano tutti gli elementi, a partire dalla posizione iniziale fino a quella finale, effettuando i confronti e gli eventuali scambi.
Ciascuna di queste scansioni dovrà però essere a sua volta ripetuta, quindi avremo un altro blocco ripetizione (i due blocchi risultano annidati).

Cosa cambia fra una scansione completa e la successiva? Si modificano le dimensioni della parte di vettore già ordinato e di quella ancora da ordinare. Più specificatamente si sposterà l'indicatore che abbiamo chiamato INIZIO, che rappresenta la posizione iniziale del vettore ancora da ordinare. Inizio parte dalla posizione zero, e man mano si sposta verso la fine.

[61] Un metodo di ordinamento simile a quello qui illustrato è noto con il nome di "bubble sort" ad indicare proprio i valori più piccoli che *salgono* verso le prime posizioni come fossero delle bolle.

Osserva e leggi il codice corrispondente




```
using System;
   class Ordinamento
   {
      static void Main()
      {
         int[] v = { 12,88,3,45,67,25,86,46,72,95,5,32};
         int FINE = 12;
         int xyz;
         //visualizzo il vettore iniziale
         Console.WriteLine("vettore iniziale:    ");
         for (int x = 0; x < FINE; x++)
            Console.Write("{0}   ", v[x]);
         Console.WriteLine();

         Console.WriteLine("\nordinamento in corso .. . . ");
         for (int INIZIO = 0; INIZIO < FINE-1; INIZIO++)
         {
            for (int I = INIZIO + 1; I < FINE; I++)
            {
               if (v[INIZIO] > v[I])
               {  //scambio
                  xyz = v[INIZIO];
                  v[INIZIO] = v[I];
                  v[I] = xyz;
               }
            }
            //visualizzo il vettore dopo ogni scansione
            for (int x = 0; x < 12; x++)
               Console.Write("{0}   ", v[x]);
            Console.WriteLine();
         }
         Console.ReadLine();
      }
   }
```

4.4. MATRICI (ARRAY A DUE DIMENSIONI)

PREREQUISITI

array ad una dimensione

OBIETTIVI

Conoscere il tipo di dati array a due dimensioni ed utilizzarlo in modo adeguato per la soluzione di problemi

4.4.1. Matrici

4.4.1.1. Un problema da risolvere

Problema→ Un gruppo di 4 alunni esamina i voti che ciascuno di essi ha ottenuto in tre diverse verifiche scritte di informatica, come riportato nello schema: in particolare confrontano le media ottenute da ciascuno di essi e le medie per ciascuna prova. L'insegnante propone di sviluppare una applicazione per gestire questi dati.

Luigi	**5**	**6**	**7**
Elisa	**4**	**8**	**5**
Gianni	**9**	**6**	**5**
Ugo	**5**	**6**	**7**

Analisi

Le elaborazioni da effettuare sono semplici, ma ci sono discussioni su come memorizzare i dati. Per i nomi si pensa di utilizzare un vettore di stringhe, ma per i voti i sono due idee: tre vettori (corrispondenti alle colonne della tabella) con i voti di ogni studente in una singola prova, oppure 4 vettori (corrispondenti alle righe della tabella), ciascuno dei quali raccoglie i voti di ogni singolo alunno. Ciascuna delle due proposte ha punti di forza e di debolezza: i dati sono tutti dello stesso tipo, si potrebbe addirittura pensare ad un vettore di 12 elementi, ma la realtà a cui i dati si riferiscono rende significativo il rappresentarli con uno schema a due dimensioni, le colonne che rappresentano le prove, le righe che si riferiscono invece agli alunni. La soluzione migliore è quindi quella di utilizzare un array a due dimensioni, che viene detto matrice. Si lascia al lettore il compito di completare l'analisi pensando in particolare a come sviluppare le diverse funzionalità per la gestione dei dati.

Tabella di descrizione delle variabili[62]

Nome	Descrizione	Tipo di dato
N [4]	Nomi degli studenti	Vettore di stringhe
M [4, 3]	Matrice con i voti degli studenti	Matrice di interi
R	Contatore che individua la riga	intero
C	Contatore che individua la colonna	intero

[62] Si indicano qui solo le variabili per gestire i dati del problema

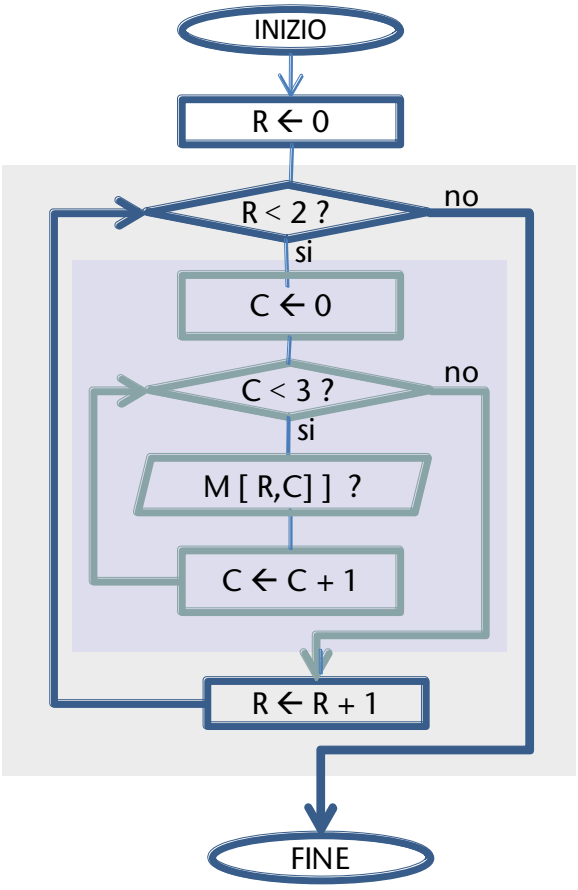
Algoritmo memorizzazione voti riga per riga

Si presenta qui solo la parte di algoritmo relativa alla memorizzazione dei voti, scegliendo di memorizzare prima tutti i dati del primo alunno, di seguito quelli del secondo alunno , e così via. Il primo dato dovrà essere memorizzato nell'elemento in alto a sinistra e, come per i vettori, si dovrà individuare tale elemento indicando la **posizione**, che in questo caso sarà data **specificando due coordinate**, prima quella della **riga**, poi quella della **colonna** (si inizia sempre contare da zero).

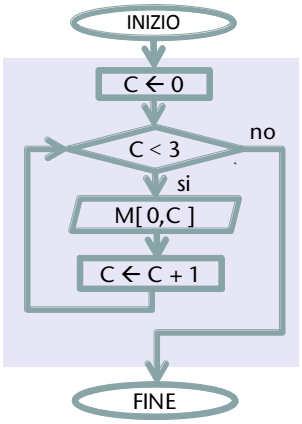
	0	1	2
0	5	6	7
1	4	8	5
2	9	6	5
3	5	6	7

Lo schema a lato indica i valori della matrice e le posizioni degli elementi: ad esempio il valore 8 si trova nella posizione [1,1], il valore 9 in [2,0].

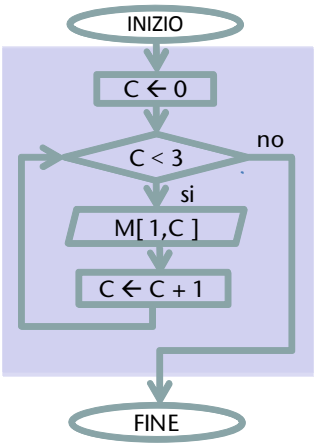
Algoritmo memorizzazione dati matrice



Algoritmo memorizzazione dati riga 0



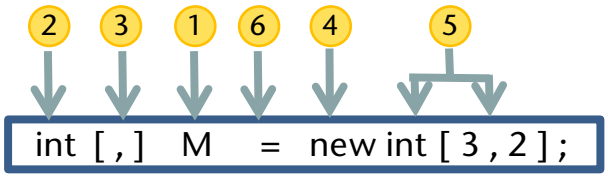
Algoritmo memorizzazione dati riga 1



Da osservare che i due algoritmi a lato sono assolutamente identici, tranne per l'indicazione del numero della riga in cui memorizzare i dati: invece di ripetere tali algoritmi per ogni riga, si è racchiuso l'algoritmo stesso in un blocco ripetizione esterno che fa variare l'indicatore della riga.

4.4.1.2. Dichiarazione e inizializzazione di una matrice

Una **matrice** (array a due dimensioni) è una **collezione di dati dello stesso tipo, ciascuno identificato dalla posizione** che viene individuata specificando le coordinate della **riga** e della **colonna**.



1	M	Identificatore, cioè nome della matrice
2	Int	Tipo di dato degli elementi della matrice
3	[,]	Operatore di indicizzazione: mostra che M è una matrice
4	New	Operatore: alloca lo spazio per gli elementi della matrice: questo indica che la matrice è una variabile di tipo riferimento, come il vettore
5	int [3 , 2]	Indica che la matrice è costituito di elementi di tipo intero organizzati secondo una tabella costituita d 3 righe e 2 colonne
6	=	Operatore: assegna a M il riferimento agli elementi allocati in memoria

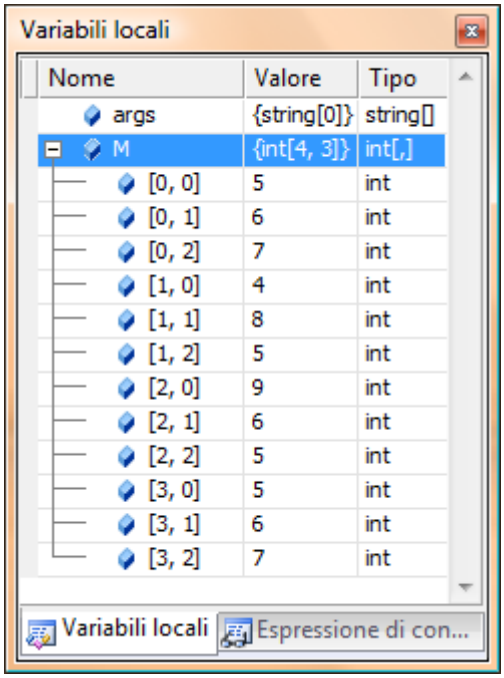
Altri esempi di dichiarazione di matrici possono essere i seguenti:

```
int righe=3, colonne=5;
int [,] tabella = new int [righe,colonne];

double[,] Matrice;
Matrice = new double[4,2];
```

È possibile inizializzare una matrice; ad esempio per ottenere i valori dell' esempio precedente si scrive:

```
int[,] M = { { 5, 6, 7 }, { 4, 8, 5 }, { 9, 6, 5 }, { 5, 6, 7 } };
```

Osserva cha la matrice è considerata come se fosse un *vettore di vettori*, ciascuno dei quali corrisponde ad una riga.



4.4.2. Utilizzare le matrici

Vengono mostrati alcuni esercizi con matrici, quindi un esempio di applicazione per la soluzione di un semplice problema con l'uso di vettori e matrici per la gestione dei dati.

4.4.2.1. Elaborazione per riga

Problema→ I 4 alunni indicati nel paragrafo 4.4.1.1 stanno confrontando i loro voti: si richiede di calcolare la media di ciascuno di essi.

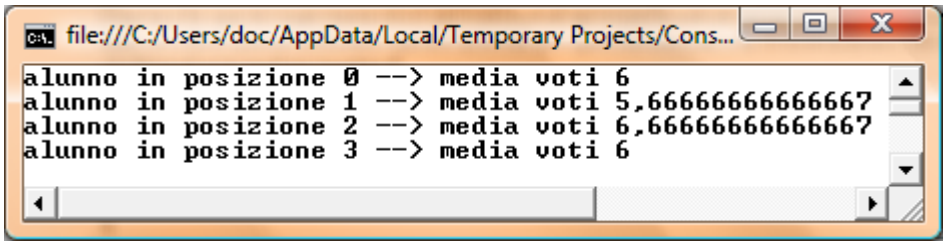
Analisi

Proviamo a pensare: se vogliamo calcolare la media del primo alunno dobbiamo sommare i suoi voti e dividerli per tre: i voti si trovano sulla prima riga della tabella, che risulta essere a tutti gli effetti un vettore, quindi il calcolo della media sarà analogo a quello già presentato (un blocco ripetizione).
Siccome dobbiamo eseguire tale calcolo per ciascuno degli alunni, dovremo di seguito esaminare i quattro vettori che corrispondono ad ogni riga della matrice.

Due possibilità nella realizzazione dell' algoritmo: quattro blocchi ripetizione in sequenza, ciascuno riferito ad una riga, oppure un solo blocco ripetizione con l' indicazione che deve essere ripetuto 4 volte (cioè il blocco ripetizione per l'elaborazione di una singola riga *annidato* in un altro che fa sì che venga ripetuto 4 volte, cambiando man mano il riferimento alla riga da elaborare). Naturalmente scegliamo la seconda soluzione che è di carattere più generale.

Osserva che questo ragionamento è del tutto analogo a quello già proposto per la memorizzazione dei dati: in effetti potrai utilizzare questo schema per *ogni elaborazione* che ti trovi a dover effettuare sugli elementi di una *qualsiasi matrice*.

Per la tabella delle variabili considera quella presentata nei paragrafi precedenti.

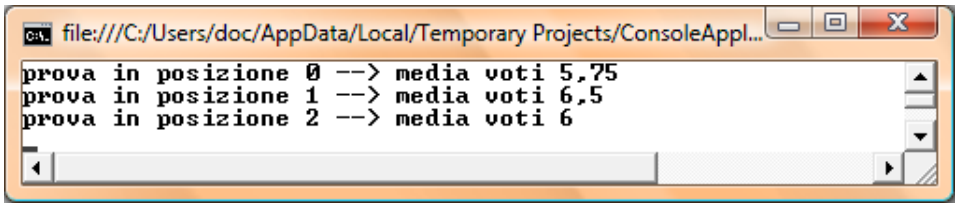



```
using System;
class Program
{
   static void Main( )
   {
      int[,] M = { { 5, 6, 7 }, { 4, 8, 5 }, { 9, 6, 5 }, { 5, 6, 7 } };
      int R, C;
      int S; //somma voti di ogni singolo alunno
      double Media;  //media voti di ogni singolo alunno
      for (R=0; R<4; R++)  //per ogni riga della matrice
      {
         S = 0; //inizio l' elaborazione dei dati dell' alunno nella riga R
         for (C = 0; C < 3; C++)
         {
            S = S + M[R, C];
         }
         Media = S / 3.0;
         Console.WriteLine ("alunno in posizione {0} --> media voti {1}",R,Media);
      }
      Console.ReadLine();
   }
}
```

4.4.2.2. Elaborazione per colonna

Problema→ I 4 alunni indicati nel paragrafo 4.4.1.1 stanno confrontando i loro voti: si richiede di calcolare la media per ciascuna prova.

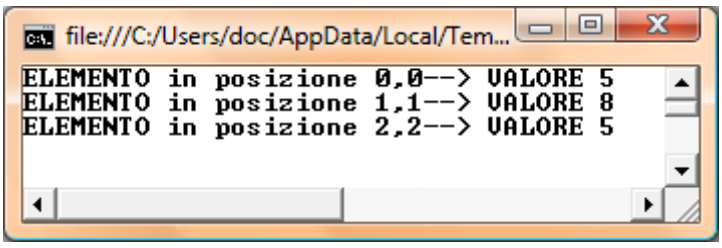
Analisi
La situazione è analoga a quella vista nel paragrafo precedente: per calcolare la media della prima prova si elaborano i dati della prima colonna, che è a tutti gli effetti un vettore, quindi si esaminano man mano le altre colonne.




```
using System;
class Program
{   static void Main( )
    {
       int[,] M = { { 5, 6, 7 }, { 4, 8, 5 }, { 9, 6, 5 }, { 5, 6, 7 } };
       int R, C;
       int S; //somma voti di ogni singolo alunno
       double Media;  //media voti di ogni singolo alunno
       for (C = 0; C < 3; C++)  //per ogni colonna della matrice
       {
          S = 0; //inizio l' elaborazione dei dati della prova nella colonna C
          for (R = 0; R < 4; R++)
          {
             S = S + M[R, C];
          }
          Media = S / 4.0;
          Console.WriteLine("prova in posizione {0} --> media voti {1}", C, Media);
       }
       Console.ReadLine();
    }
}
```

4.4.2.3. Elaborazione per diagonale

Per elaborare gli elementi sulla diagonale di una matrice quadrata osserva che anch'essi possono essere visti come un vettore (sono disposti in linea!).




```
using System;
class Program
{
   static void Main( )
   {      int[,] M = { { 5, 6, 7 }, { 4, 8, 5 }, { 9, 6, 5 }};
      int I; //posizione
      for (I = 0; I < 3; I++)  //per ogni elemanto della diagonale
      {
         Console.WriteLine("ELEMENTO in posizione {0},{1}--> VALORE {2}", I,I, M[I,I]);
      }
      Console.ReadLine();
   }
}
```

4.5. RECORD

PREREQUISITI

[dati di tipo semplice]

OBIETTIVI

[conoscere il tipo di dati record ed utilizzarlo in modo adeguato per la soluzione di problemi]

4.5.1. Record

4.5.1.1. Un problema da risolvere

Problema → Si vogliono memorizzare i dati relativi ad un alunno, in particolare interessa conoscere nominativo, residenza, media dei voti. Si prevede che possa in seguito interessare gestire dati analoghi anche per altri alunni.

Individuare la struttura più idonea per gestire tali dati.

Questo esempio presenta una situazione su cui non ci si è mai soffermati: i dati si riferiscono tutti ad una precisa entità, lo studente, quindi sarebbe interessante riuscire a conservare questo legame logico.

Si tratta di cercare di memorizzarli tutti in una sola variabile strutturata in modo adeguato, ma l'unico esempio che conosciamo di variabile strutturata è costituito dagli array, dove i dati sono tutti dello stresso tipo, mentre qui si ha a che fare con tipi differenti (nome e residenza sono stringhe, la media dei voti è un valore numerico).

Un altro aspetto nuovo: fin qui abbiamo utilizzato tipi di dati predefiniti (interi, double, char, stringhe, boolean) di uso assai generale; ora invece ci servirebbe poter utilizzare un tipo di dato molto specifico, che riuscisse a rappresentare la nostra particolare situazione, il nostro alunno : è chiaro che ben difficilmente può esistere predefinito proprio il tipo di dati che ci interessa.

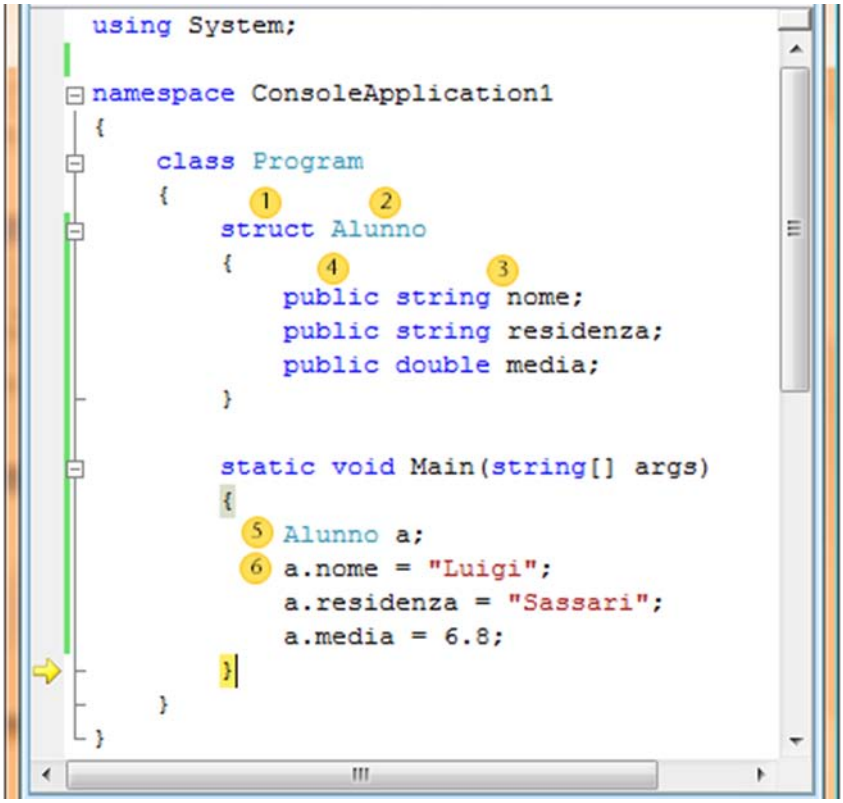
Da qui in avanti saremo noi a definire nuovi tipi di dato!

4.5.1.2. Definizione di un record

Un **record** è una collezione di dati in genere di tipi differenti, ciascuno identificato dal nome. Ogni elemento viene detto **campo** del record ed è caratterizzato da un nome e da un tipo; ogni elemento può essere di tipo semplice o, a sua volta, di tipo strutturato.
La definizione di un nuovo tipo di dato non comporta allocazione di memoria, ma rende possibile dichiarare una variabile di quel tipo: sarà la variabile corrispondente a riservare in memoria lo spazio per registrare i dati.

Strutture e classi
Le classi e le strutture vengono utilizzate per la definizione di nuovi tipi di dati. Una struttura può essere considerata una classe leggera, ideale per creare tipi di dati che archiviano piccole quantità di dati.
Dal manuale MSDN

Per la definizione dei record viene utilizzata nel linguaggio C# la parola chiave struct: una **struttura** è associata ad una variabile di tipo valore, in quanto nella variabile vengono memorizzati direttamente i dati. Di seguito un esempio.

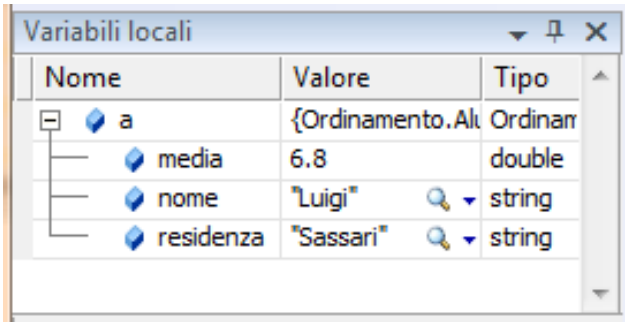


Codifica C# 4-1 - esempio di struttura

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

1	struct	La parola chiave struct indica che si sta procedendo alla definizione di una struttura
2	Alunno	Identificatore della struttura, specifica l'entità che si sta considerando
3	string nome	Questo è il primo campo della struttura, si specifica tipo e identificatore del campo, di seguito si indicano nello stesso modo tutti gli altri campi
4	public	Parola chiave, indica che questo identificatore è pubblico, cioè potrà essere utilizzato fuori dal blocco delimitato dalle parentesi graffe[63]
5	Alunno a	Dichiaro una variabile a di tipo Alunno: qui viene allocato lo spazio in memoria
6	a.nome = "Luigi"	Assegno il valore Luigi al campo nome della variabile a: osserva l'uso dell'operatore punto per accedere al campo del record

[63] Questa è una spiegazione un po' superficiale, sarà più chiaro in seguito, quando si tratterà la programmazione ad oggetti OOP, volume secondo



4.5.1.3. Utilizzare i record per gestire un tabella

Riprendiamo l' esempio relativo alla gestione di nomi e voti alunni (vedi paragrafo 4.3.2.2): invece di utilizzate due vettori per gestire i nomi ed i voti, ora possiamo considerare l' entità alunno quindi, per gestire i dati dell' elenco, sarà sufficiente un vettore di alunni.

Analisi
La soluzione proposta qui, analoga a quella già discussa dal punto di vista logico, è però migliore rispetto alla gestione dei dati.

Tabella di descrizione del record Alunno

Nome campo	Descrizione	tipo
nome	nominativo dell' alunno	stringa
voto	voto dell' alunno	intero

Tabella di descrizione delle variabili

Nome variabile	Descrizione	tipo	Input/Output
V[5]	Vettore di alunni	record alunno	Input/Output
B	Contatore numero. studenti Bravi, con voto almeno sufficiente	intero	output
i	Contatore, indica la posizione dell'elemento che si esamina	intero	lavoro

La tabella seguente mostrano lo stesso esempio di possibili valori già esaminato in precedenza.

	nome	voto
V[0]	giuseppe	4
V[1]	elisa	7
V[2]	maria	8
V[3]	matteo	8
V[4]	aldo	5

Algoritmo

Vedi la seconda versione dell'algoritmo di paragrafo 4.3.2.2

Codifica

```
using System;
  class Ordinamento
  {
     struct Alunno
     {
        public string nome;
        public int voto;
     }

     static void Main()
     {
        Alunno[ ] V = new Alunno[5];
        int i;        //contatore
        int B = 0; // contatore numero bravi
        int sc;   //scelta opzione menu
        string tmp;

        do
        {
           // presentazione menu
           Console.WriteLine("1. Inserimento nominativi alunni");
           Console.WriteLine("2. Inserimento voti alunni");
           Console.WriteLine("3. Visualizza dati");
           Console.WriteLine("4. Conta alunni bravi");
           Console.WriteLine("5. Elenco alunni recupero");
           Console.WriteLine("0. Esci");
           Console.WriteLine("Indica la tua scelta");

           tmp = Console.ReadLine();
           sc = Convert.ToInt32(tmp);

           switch (sc)
           {
              case 1:
                 //inserimento nomi
                 Console.WriteLine("Inserimento nominativi alunni");
                 for (i = 0; i < 5; i++)
                 {
                    Console.WriteLine("dammi il nominativo dell' alunno numero {0}: ", i + 1);
                    V[i].nome = Console.ReadLine();
                 }
                 break;

              case 2:
                 //inserimento voti
                 Console.WriteLine("\nInserimento voti alunni");
                 for (i = 0; i < 5; i++)
                 {
                    Console.WriteLine("dammi il voto dell' alunno  0}: ", V[i].nome);
                    tmp = Console.ReadLine();
                    V[i].voto = Convert.ToInt32(tmp);
                 }
                 break;
```

```
            case 3:
               //visualizzazione dati
               Console.WriteLine("\nTabella dati alunni");
               Console.WriteLine("nomi\tvoti");
               for (i = 0; i < 5; i++)
               {
                  Console.WriteLine(V[i].nome + "\t" + V[i].voto);
               }
               break;

            case 4:
               //conteggio alunni bravi
               for (i = 0; i < 5; i++)
               {
                  if (V[i].voto >= 6)
                     B = B + 1;
               }
               Console.WriteLine("\nTotale alunni bravi {0}: ", B);
               break;

            case 5:
               //elenco alunni recupero
               Console.WriteLine("\nElenco alunni recupero");
               for (i = 0; i < 5; i++)
               {
                  if (V[i].voto < 6)
                     Console.WriteLine(V[i].nome);
               }
               break;
         } //fine switch
      } while (sc != 0);

      Console.WriteLine("\nPremi INVIO per chiudere il programma");

      Console.ReadLine();
   }//fine Main
}//fine class
```

Output

L' output prodotto è mostrato al paragrafo 4.3.2.2: questa soluzione non modifica l'interfaccia con l'utente, la differenza con il caso precedente consiste nell'utilizzo di una tecnica più avanzata nella gestione dei dati.

4.6. FILE DI TESTO

PREREQUISITI

Conoscere i dati di tipo semplice]

OBIETTIVI

Saper memorizzare i dati di una applicazione su file

4.6.1. File

È importante acquisire le competenze essenziali per memorizzare dati su disco: a ciò è dedicato il presente capitolo.

4.6.1.1. Un problema da risolvere

Problema→ L'insegnante di informatica intende iniziare oggi una serie di interrogazioni che si dovrebbero concludere entro due settimane circa, ed avrebbe la necessità di avere un elenco degli alunni già sentiti registrando i nominativi al termine di ogni colloquio.

Come altre volte il problema da risolvere sembra abbastanza semplice ma nasconde una difficoltà: potremmo registrare i nominativi utilizzando un vettore di stringhe, ma al termine dell'esecuzione del programma tutti i dati memorizzati nella RAM si perdono. Alcune soluzioni: chiedere all'insegnante di non spegnere il computer e di non chiudere l'applicazione sino al termine delle interrogazioni; oppure: imparare a gestire i dati memorizzati su disco!

4.6.1.2. File di testo

I **file** sono sequenze di byte su supporti di memorizzazione di massa che consentono di immagazzinare grandi quantità di dati in modo permanente.

File di testo sono file in cui i dati sono memorizzati come sequenze di caratteri, cioè rappresentano un testo come una serie di caratteri Unicode.

Operazioni

- Apertura: specificare la modalità lettura, scrittura, accoda
- Lettura: accesso sequenziale
- Scrittura: accesso sequenziale
- Chiusura

4.6.1.3. Gestione file di testo in C#

Leggere e scrivere su un file di testo non è dissimile dal leggere da tastiera e scrivere a video.
Consideriamo la scrittura: si tratta di trasferire dati che sono memorizzati nella RAM su un supporto esterno, utilizzando un canale di comunicazione adeguato.
Conosciamo un modo (in termini tecnici: un metodo) per scrivere a video? Certo: il metodo WriteLine, riferito alla Console, ad esempio:

```
Console. WriteLine("Ciao");
```

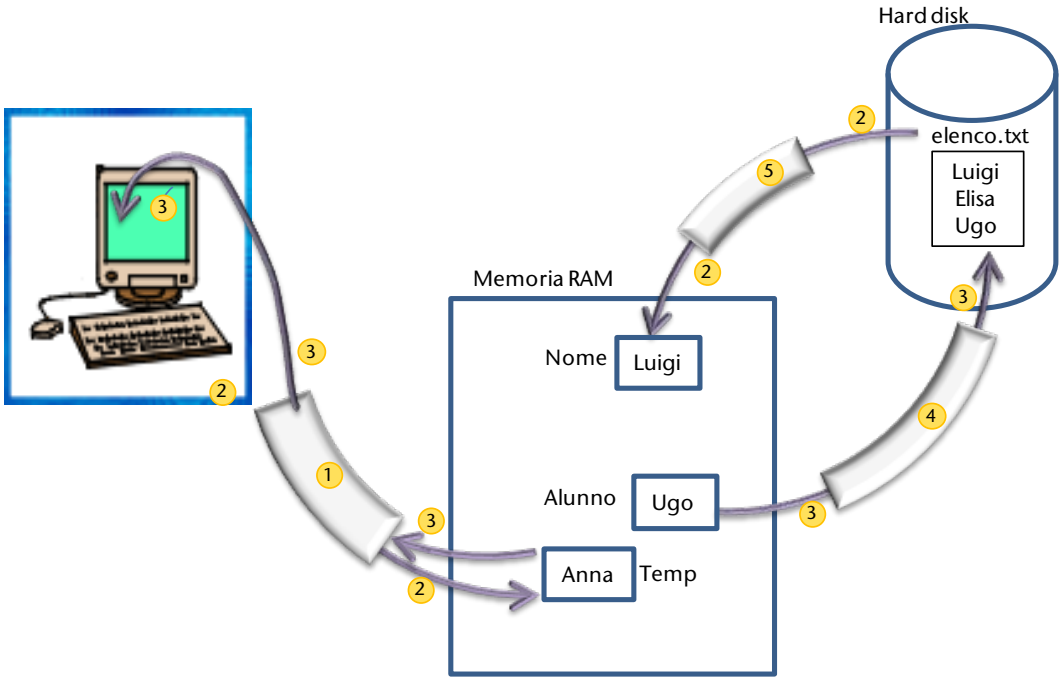
Supponiamo ora di avere un collegamento al file di testo d:\elenco.txt memorizzato su disco, tale collegamento si può chiamare ad esempio MioFile: riesci ad immaginare quale può essere il significato della seguente istruzione?

```
MioFile.WriteLine("Ciao");
```

La parola Ciao verrà scritta su disco, nel file collegato a MioFile, cioè nel file d:\elenco.txt.

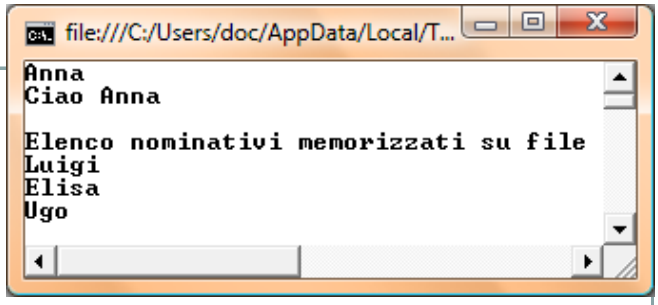
Il collegamento con la Console è stabilito direttamente dal sistema, ora sarà sufficiente imparare a stabilire i collegamenti con i nostri file; per il resto possiamo direttamente utilizzare competenze che abbiamo già acquisito in questo nuovo contesto, scrivendo sul file come se stessimo scrivendo sul video. Analogo discorso vale per la lettura dei dati.

Osserva le figura seguente e cerca di comprendere il significato di tutti i punti indicati, di seguito prova a leggere il listato del codice corrispondente.



1 Console	Rappresenta il collegamento con la Console
2 ReadLine( )	Lettura di una riga di dati dal canale di input e trasferimento in memoria centrale
3 WriteLine( )	Trasferimento dati dalla memoria centrale e scrittura di una riga sul canale di output
4 StreamWriter	Canale di output, cioè canale per scrivere dati su disco (output dalla RAM)
5 StreamReader	Canale di input, cioè canale per leggere dati dall'esterno (input nella RAM)

Osserva e leggi il codice corrispondente




```
1  using System;
2  using System.IO;
6          class Program
7          {
8             static void Main(string[] args)
9             {
10               string Temp, Alunno="Ugo", Nome;
11         1 2   Temp = Console.ReadLine();
12         1 3   Console.WriteLine("Ciao " + Temp);
13
14            4  StreamWriter ScriviSuFile = new StreamWriter(@"d:\C#\elenco.txt", true);
15         4  3  ScriviSuFile.WriteLine(Alunno);
16               ScriviSuFile.Close();
17
18            5  StreamReader LeggiDaFile = new StreamReader(@"d:\C#\elenco.txt");
19         5  2  Console.WriteLine("\nElenco nominativi memorizzati su file");
20               Nome = LeggiDaFile.ReadLine();
21               while (Nome != null)
22                   {
23                      Console.WriteLine(Nome);
24                      Nome = LeggiDaFile.ReadLine();
25                   }
26               LeggiDaFile.Close();
27            }
28         }
```

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 2 uso il namespace System.IO dove sono definiti StreamReader e StreamWriter

riga 11 Lettura da tastiera: il valore inserito viene memorizzato nella stringa Temp;

riga 12 scrittura a video, fra parentesi tonde si indica ciò che si vuole mandare a video;

riga 14 creo uno StreamWriter, cioè un canale di comunicazione per scrivere su file. ScriviSuFile è il nome del mio canale di comunicazione; il file cui fare riferimento è indicato fra parentesi tonde @"d:\C#\elenco.txt" il percorso completo viene rappresentato come stringa e quindi racchiuso fra apici, preceduto da @. Di seguito è indicato true: ciò significa che se il file non esiste ancora viene creato, se esiste i nuovi valori vengono aggiunti in coda;

riga 15 questa istruzione è del tutto analoga a quella di riga 12: scrittura sul file, fra parentesi tonde ciò che si vuole memorizzare, in questo caso il valore della variabile Alunno, cioè Ugo.

riga 16 chiudo il canale ScriviSuFile che consente di accedere al file in scrittura. Questo renderà possibile effettuare eventuali altri tipi di accessi allo stesso file;

riga 18 crea il canale di comunicazione LeggiDaFile che essendo di tipo StreamReader consente di leggere i valori memorizzati su disco e trasferirli nella RAM; il file cui ci si riferisce è quello cu cui prima si era aggiunto il nome Ugo;

riga 20 Leggo una riga, non da tastiera, ma dal canale che ho aperto nella istruzione precedente: verrà memorizzato sulla variabile Nome il dato della prima riga del file, in questo caso il nome Luigi e ci si sposta alla riga successiva; quando si raggiunge la file del file viene restituito un riferimento null.

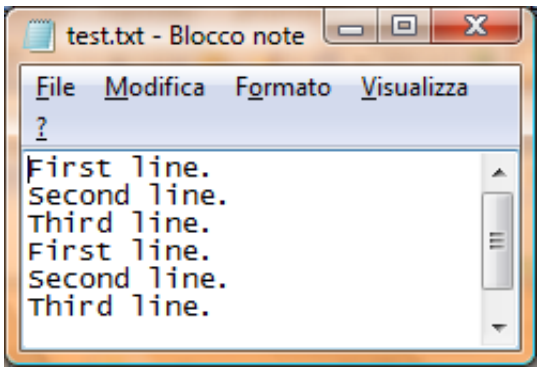
riga 21 Quando leggendo dal file (riga 24) viene restituito null si riconosce la fine del file stesso.

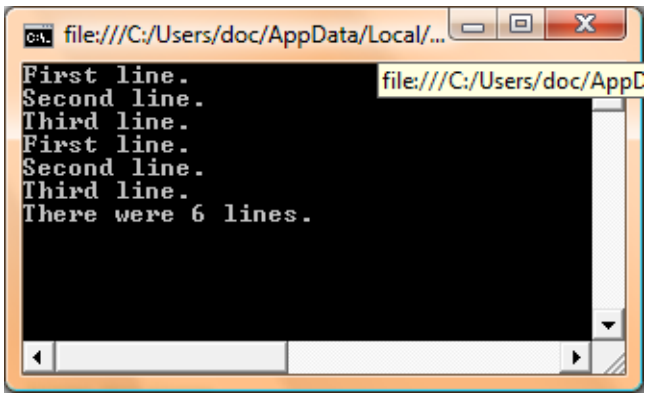
4.6.2. Utilizzare i file di testo

Vengono mostrati semplici esempio di applicazioni con l'uso di file di testo, per i dettagli vedi manuale del linguaggio nell'Help di Visual C# o di SharpDeveloper. .

4.6.2.1. Contare il numero di righe di un file di testo

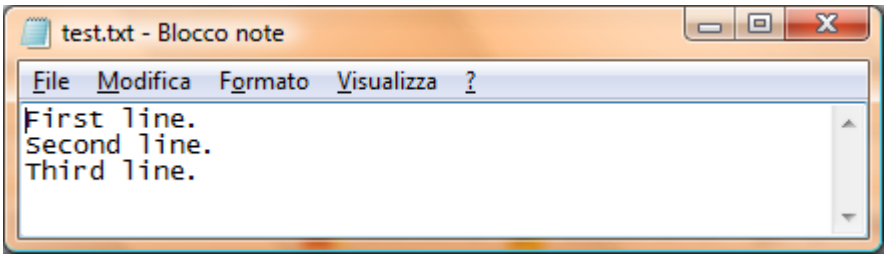
```
using System;
using System.IO;

namespace Prova
{
   class Esercizio
   {
      static void Main()
      {
         int contatore = 0; // numero di linee del testo
         string line;

         // legge il file e lo visualizza riga per riga
         StreamReader file = new StreamReader(@"d:\c#\test.txt");
          line = file.ReadLine(); //leggo la prima riga del file e mi posiziono sulla successiva
          while (line != null)
          {
            Console.WriteLine(line);
            contatore++;
            line = file.ReadLine(); //leggo una riga del file e mi posiziono sulla successiva
                                    // quando raggiungo la fine del file la variabile line assume il valore null
         }
         file.Close();
         Console.WriteLine("Numero totale linee :  {0}.", contatore);

         Console.ReadLine();
      }
   }
}
```





4.6.2.2. Scrivere su un file di testo

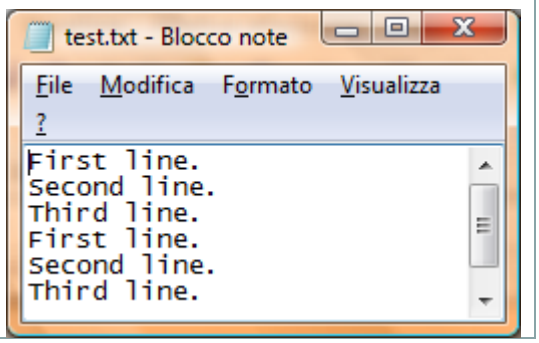
```
                                        using System;
using System.IO;

// Se il file non esiste, ne viene creato uno nuovo. Se esiste, viene sovrascritto.

namespace Prova
{
   class Esercizio
   {
      static void Main()
      {
         // costruisco una stringa formata da tre linee di testo.
         string lines = "First line.\r\nSecond line.\r\nThird line.";

         // Scrivo la stringa sul file
         StreamWriter file = new StreamWriter(@"d:\C#\test.txt");
         file.WriteLine(lines);

         file.Close();

      }
   }
}
```



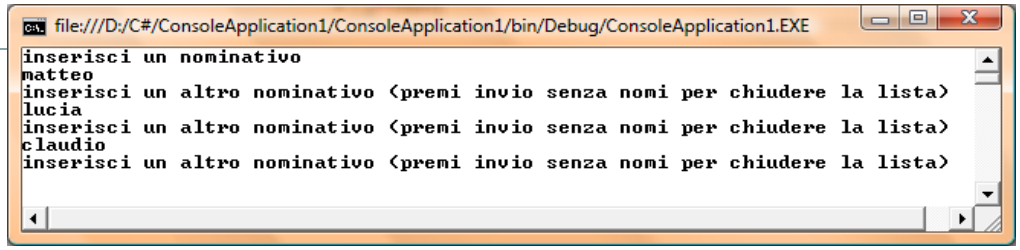
In questi esempi si utilizzano i parametri in modo intuitivo, senza conoscere la teoria (anche chi guida l'automobile probabilmente non sa come funziona un motore!). Vedi il prossimo capitolo per una spiegazione completa.

4.6.2.3. Aggiungere elementi ad un file di testo

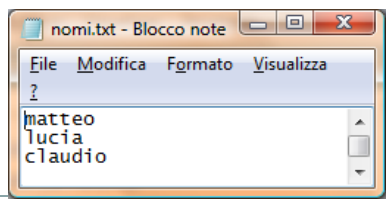
```
using System;
using System.IO;
// Se il file non esiste, ne viene creato uno nuovo. Se esiste, i nuovi valori vengono aggiunti in coda.

namespace Prova
{   class Esercizio
   {  static void Main()
      {
         // costruisco una stringa formata da tre linee di testo.
         string lines = "First line.\r\nSecond line.\r\nThird line.";

         // Scrivo la stringa sul file.
         StreamWriter file = new StreamWriter(@"d:\C#\test.txt",true);
         file.WriteLine(lines);

         file.Close();

      }
   }
}
```



Un secondo esempio

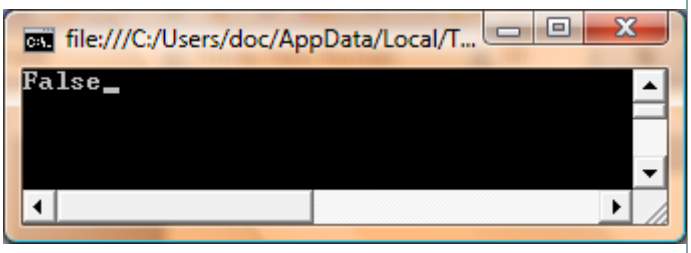



```
using System;
using System.IO;
namespace prova
{
   class FileTesto
   {
      static void Main()
      {  string tmp;
         StreamWriter sw = new StreamWriter(@"d:\c#\file\nomi.txt");
         Console.WriteLine("inserisci un nominativo");
         tmp = Console.ReadLine();
         while (tmp != "")
         {
            sw.WriteLine(tmp);
            Console.WriteLine("inserisci un altro nominativo (invio senza nomi per chiudere la lista)");
            tmp = Console.ReadLine();
         }
         sw.Close();
      }
   }
}
```



4.6.2.4. Verificare l' esistenza di un file

```
using System;
using System.IO;


namespace Prova
{
   class Esercizio
   {
      static void Main()
      {
         bool FileEsiste;
         FileEsiste = File.Exists(@"c:\myfile.txt");
         System.Console.Write(FileEsiste);
         Console.ReadLine();
      }
   }
}
```



HAI IMPARATO A ...

1. Comprendere l' importanza dei tipi di dati
2. Utilizzare i tipi int, double, char, string, boolean, record
3. Utilizzare i tipi array (vettori e matrici)
4. Utilizzare i file di testo

5. Scomposizione in sottoprogrammi

Questa sezione introduce all'uso della metodologia top-down per la soluzione di problemi e presenta le relative tecniche di implementazione in linguaggio C# con la scomposizione del codice in sottoprogrammi (metodi di classe) e l'utilizzo dei parametri.

- Metodologia top-down per la soluzione problemi complessi
- Sviluppo di applicazioni

5.1. METODOLOGIA TOP-DOWN PER LA SOLUZIONE PROBLEMI COMPLESSI

PREREQUISITI

Saper risolvere problemi semplici

OBIETTIVI

Saper risolvere problemi complessi

5.1.1. Struttura di un programma con l'uso di metodi

Per sviluppare abilità nella soluzione dei problemi è necessario acquisire un buon metodo di lavoro: nel caso di situazioni complesse è fondamentale suddividere il problema in sottoproblemi più semplici e quindi, in fase di codifica in linguaggio C#, strutturare il programma con l'uso di metodi e parametri.

5.1.1.1. Un problema da risolvere

In un ufficio lavorano alcuni impiegati, ciascuno dei quali ha una propria area di lavoro e svolge dei compiti ben precisi. Vi è inoltre la figura di un responsabile dell'ufficio il cui ruolo è quello di custodire i documenti e di coordinare il lavoro dei diversi impiegati.

Un'impresa edile ha ricevuto lo schema relativo al lavoro che sta svolgendo in un cantiere per l'installazione di un caminetto prefabbricato, con l'indicazione di tutte le misure.

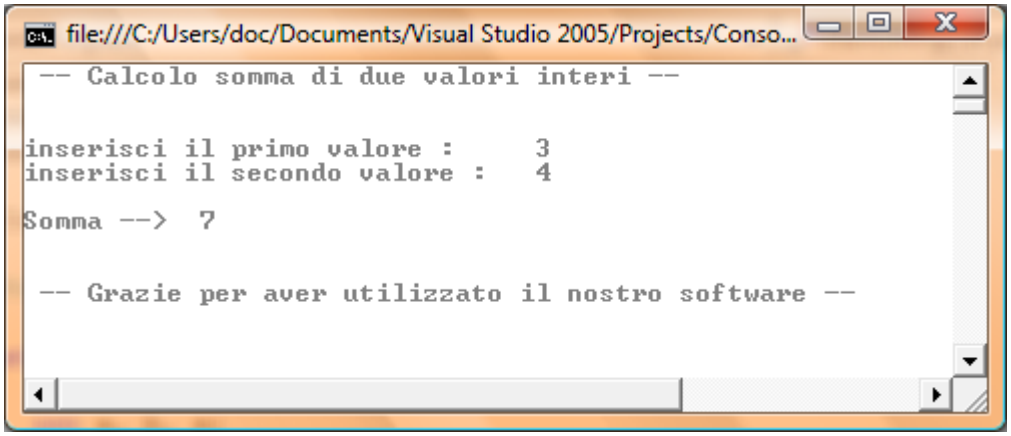
Cosa hanno in comune le situazioni proposte? Si tratta di casi complessi gestiti attraverso la scomposizione del problema generale in "parti" più semplici.

La complessità del lavoro dell'ufficio è risolta assegnando a ciascun impiegato il proprio ambiente di lavoro (scrivania ad esempio) ed un compito preciso; l'impresa edile non deve progettare e realizzare il caminetto, ma lavora utilizzando oggetti prefabbricati reperibili sul mercato o presso le aziende costruttrici.

È però in tutti i casi necessario prestare attenzione affinché venga garantito il buon funzionamento del sistema nel suo complesso.

Il lavoro dei singoli impiegati viene coordinato dal responsabile dell'ufficio; l'impresa edile deve attenersi con precisione alle specifiche tecniche fornite dai fornitori dell'oggetto da utilizzare.

Osserva e leggi il codice corrispondente



5-1 *Finestra di output programma per il calcolo della somma di due valori interi*

```
using System;
class Program
{
   static void Main( )
   {
      int a, b, s;
      string tmp;
      Console.WriteLine("-- Calcolo somma di due vaori interi --\n\n");
      Console.Write("inserisci il primo valore:\t");
      tmp = Console.ReadLine();
      a= Convert.ToInt32 (tmp);
      Console.Write("inserisci il secondo valore:\t");
      tmp = Console.ReadLine();
      b= Convert.ToInt32 (tmp);
      s = a + b;
      Console.WriteLine("\nSomma --> " + s);

      Console.WriteLine("\n--Grazie per aver utilizzato il nostro software --");
      Console.ReadLine();

   }

}
```

Il codice proposto si riferisce ad un esempio molto semplice: esso non presenta le caratteristiche di scomposizione viste negli esempi precedenti; nel capitolo seguente proviamo a modificare il nostro lavoro in modo da utilizzare nel nostro campo, lo sviluppo del software, le stesse tecniche che abbiamo apprezzato come vantaggiose nei casi osservati.

5.1.1.2. Classe con più metodi

Leggi il codice

```
1  using System;
2  class Program
3  {
4
5     static void Main( )
6     {
7        Presentazione( ); //chiamata del metodo
8        Elaborazione( ); //chiamata del metodo
9        Console.WriteLine("\n--Grazie per aver utilizzato il nostro software --");
10       Console.ReadLine();
11    }
12
13    //definizione metodo Presentazione
14    static void Presentazione( )
15    {
16       Console.WriteLine("-- Calcolo somma di due vaori interi --\n\n");
17    }
18
19    //definizione metodo Elaborazione
20    static void Elaborazione( )
21    {
22       int a, b, s;
23       string tmp;
24       Console.Write("inserisci il primo valore:\t");
25       tmp = Console.ReadLine();
26       a= Convert.ToInt32 (tmp);
27       Console.Write("inserisci il secondo valore:\t");
28       tmp = Console.ReadLine();
29       b= Convert.ToInt32 (tmp);
30       s = a + b;
31       Console.WriteLine("\nSomma --> " + s);
32    }
33 }
```

Vediamo una classe costituita da tre differenti moduli che, per utilizzare il linguaggio della programmazione ad oggetti, vengono propriamente chiamati **metodi**. Il punto di ingresso del programma[64] è il metodo **Main**, il quale però non contiene le istruzioni relative alle diverse attività da eseguire, ma **richiama i metodi** che devono svolgere i compiti specifici.

[64] Cioè il punto da cui ha inizio l'esecuzione del programma

Il comportamento del Main è analogo a quello del responsabile dell'ufficio, che demanda ai diversi impiegati lo svolgimento delle funzioni di cui essi sono responsabili, assicurando comunque il coordinamento generale.

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 7	contiene la prima istruzione che viene eseguita, con la chiamata del metodo Presentazione;
riga 14:	contiene l'intestazione del metodo Presentazione che è stato richiamato: ora verranno eseguite le istruzioni del corpo di tale metodo (si ricorda che corpo di un metodo è il blocco di istruzioni racchiuse fra le parentesi graffe che seguono l'intestazione, vedi paragrafo 3.3.1.1) ;
riga 16:	visualizza il messaggio "-- Calcolo somma di due valori interi --"; non ci sono altre istruzioni in questo metodo, quindi si ritorna al metodo chiamante (in questo caso il metodo Main) e verrà eseguita l'istruzione immediatamente successiva alla chiamata del metodo eseguito;
riga 8:	richiama il metodo Elaborazione;
riga 20:	contiene l'intestazione del metodo Elaborazione che è stato richiamato;
righe 22-31:	contengono le istruzioni del corpo del metodo Elaborazione, che vengono ora eseguite una di seguito all'altra;
riga 32:	chiude il blocco di istruzioni del metodo Elaborazione; di seguito verranno eseguire le rimanenti istruzioni del metodo Main;
riga 9:	visualizza il messaggio "-- Grazie per aver utilizzato il nostro software --"
riga 10:	attende la pressione di un tasto da parte dell'utente per chiudere l'applicazione.

L'output prodotto da tale programma è esattamente quello mostrato in Figura **5-1** Finestra di output programma per il calcolo della somma di due valori interi.

Abbiamo visto che è possibile suddividere il listato in metodi differenti e questo ci consente, in fase di analisi e progettazione, di scomporre un problema complesso.

5.1.1.3. Metodi e parametri

Il comportamento del Main nell'esempio proposto è solo in parte confrontabile quello del responsabile dell'ufficio, il quale non si limita a demandare ai diversi impiegati lo svolgimento delle funzioni di cui essi sono responsabili, assicurando il coordinamento generale, ma si occupa anche di custodire i diversi documenti .

Può capitare inoltre che il responsabile faccia una fotocopia di un documento e la passi ad un impiegato per mettergli a disposizione alcuni dati necessari per lo svolgimento del proprio lavoro.

Ancora: il responsabile dell'ufficio può affidare ad un impiegato A il compito di ricevere richieste provenienti dal pubblico che si rivolge allo sportello; naturalmente i dati che l'impiegato ha ricevuto dal cliente allo sportello dovranno essere consegnati al responsabile per la custodia, e per essere eventualmente passati ad altro impiegato che si occupa della elaborazione.

Infine: un impiegato C dello sportello potrebbe ricevere il compito di far conoscere al cliente che si presenta allo sportello il risultato delle elaborazioni effettuate dall'ufficio; specificatamente: un impiegato B, su richiesta del responsabile, elabora dei dati e consegna i risultati al responsabile stesso; l'altro impiegato C riceve dal responsabile copia dei dati per poterli far conoscer al cliente allo sportello.

Si può sintetizzare l'organizzazione del lavoro dell'ufficio in esame con il seguente schema:

Responsabile dell'ufficio

Attività: conservare i documenti e coordinare il lavoro, chiamando i diversi impiegati ogni volta che c'è da svolgere una attività specifica

Nome dell'impiegato: A

Attività: ricevere i dati dal cliente allo sportello e passarli al responsabile dell'ufficio
Dati di input (cioè dati ricevuti dall'impiegato A):
dal responsabile dell'ufficio: nessuno
dal cliente: dati relativi alla richiesta effettuata
Dati di output (cioè dati consegnati dall'impiegato A):
al responsabile dell'ufficio: dati relativi alla richiesta effettuata dal cliente
al cliente: nessuno

Nome dell'impiegato: B

Attività: elaborare i dati
Dati di input (cioè dati ricevuti dall'impiegato B):
dal responsabile dell'ufficio: datti da elaborare
dal cliente: nessuno
Dati di output (cioè dati consegnati dall'impiegato B):
al responsabile dell'ufficio: risultato dell'elaborazione
al cliente: nessuno

Nome dell'impiegato: C

Attività: comunicare al cliente allo sportello i risultati
Dati di input (cioè dati ricevuti dall'impiegato c):
dal responsabile dell'ufficio: dati da comunicare al cliente
dal cliente: nessuno
Dati di output (cioè dati consegnati dall'impiegato c):
al responsabile dell'ufficio: nessuno
al cliente: dati elaborati dall'ufficio

Proviamo a strutturare la nostra applicazione in modo analogo, descrivendo alcuni metodi secondo le specifiche introdotte:

Main
Scopo: conserva i dati e coordina il lavoro, chiamando i diversi metodi ogni volta che c'è da svolgere una attività specifica

Nome del metodo: DammiValore

Scopo: ricevere un valore intero dall'utente del programma e passarlo al metodo chiamante
Dati di input(cioè dati ricevuti dal metodo):
dal metodo chiamante: nessuno
da tastiera: valore inserito dall'utente
Dati di output (cioè dati forniti o restituiti , dal metodo):
al metodo chiamante: valore inserito dall'utente
A video: nessuno

Nome del metodo: Elaborazione

Scopo: sommare due valori interi ricevuti dal metodo chiamante
Dati di input:
dal metodo chiamante: due valori interi
da tastiera: nessuno
Dati di output :
al metodo chiamante: risultato calcolato
A video: nessuno

Nome del metodo: Visualizza

Scopo: visualizzare un valore, che rappresenta una somma, ricevuto dal metodo chiamante
Dati di input:
dal metodo chiamante: valore da visualizzare
da tastiera: nessuno
Dati di output:
al metodo chiamante: nessuno
A video: valore ricevuto dal metodo chiamante

Leggi il codice

Sequenza di esecuzione istruzioni

Metodo Main	6,7
Metodo Presentazione	17, 18, 19
Metodo Main	8
Metodo DammiValore	22,23,24,25,26,27,28,29,30
Metodo Main	9
Metodo DammiValore	22,23,24,25,26,27,28,29,30
Metodo Main	10
Metodo Elaborazione	32,33,34,35,36
Metodo Main	11
Metodo Visualizza	38,39,40,41
Metodo Main	12,13

```
1  using System;
2  class Program
3  {
4
5     static void Main( )
6     {
7        int a, c, s;
8        Presentazione();
9        a = DammiValore();
10       c = DammiValore();
11       s = Elaborazione ( a , c);
12       Visualizza(s);
13       Console.WriteLine("\n--Grazie per aver utilizzato il nostro software --");
14       Console.ReadLine();
15    }
16
17    static void Presentazione()     //definizione metodo Presentazione
18    {
19       Console.WriteLine("-- Calcolo somma di due valori interi --\n\n");
20    }
21
22    static int DammiValore()    //definizione metodo DammiValore
23    {
24       int a;
25       string tmp;
26       Console.Write("inserisci un valore:\t");
27       tmp = Console.ReadLine();
28       a= Convert.ToInt32 (tmp);
29       return a;
30    }
31
32    static int Elaborazione(int x, int y)    //definizione metodo Elaborazione
33    {
34       int s = x + y;
35       return s;
36    }
37
38    static void Visualizza(int risultato)    //definizione metodo Visualizza
39    {
40       Console.WriteLine("\nSomma --> " + risultato);
41    }
43 }
```

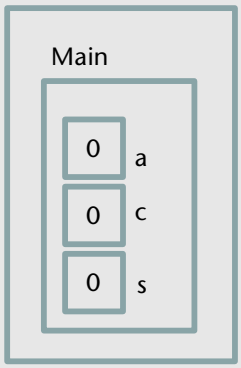
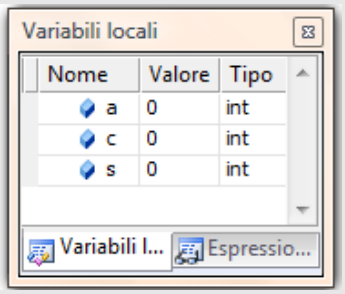
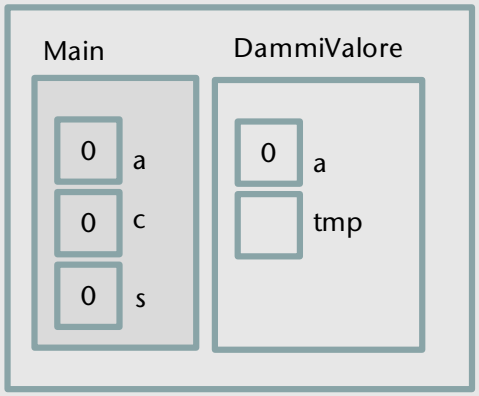
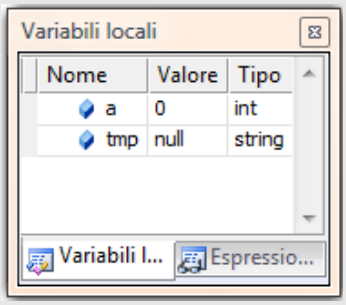
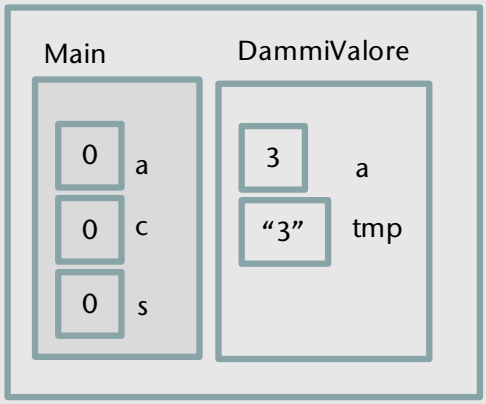
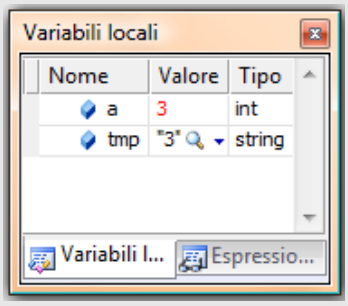
Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

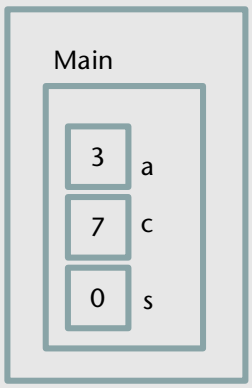
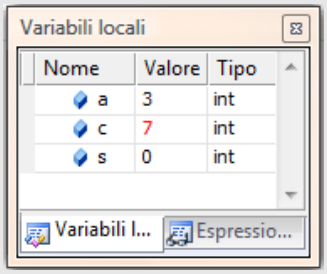
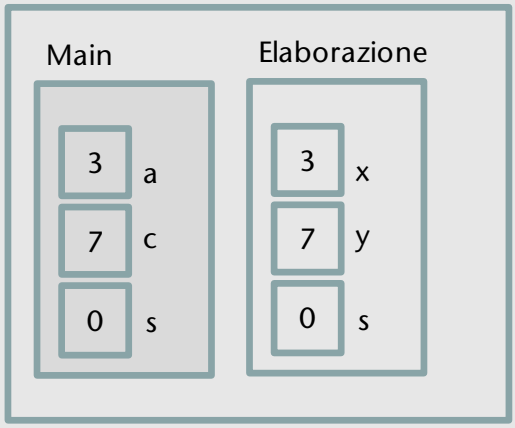
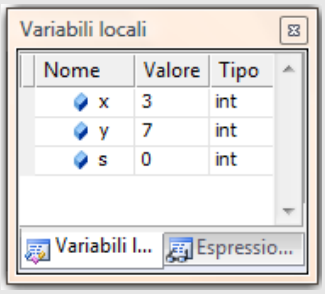
riga 7	contiene la prima istruzione che viene eseguita, con la chiamata del metodo Presentazione: viene sospesa l'esecuzione del Main, il controllo passa a Presentazione;
riga 17-20	viene eseguito il codice del metodo chiamato che visualizza il messaggio a video, quindi il controllo ritorna al Main
riga 8	chiamata del metodo DammiValore: il valore che tale metodo darà in output, cioè restituirà, al metodo chiamante (al Main in questo caso) verrà assegnato alla variabile a;
riga 22	contiene l'intestazione del metodo DammiValore che è stato richiamato; int prima del nome del metodo indica il tipo di valore restituito;
righe 24-28:	richiesta di inserimento di un valore da tastiera da parte dell'utente del programma;
riga 29	return termina l'esecuzione del metodo; return seguito dal nome di una variabile indica che il valore corrispondente viene passato (restituito) dal metodo in questione al metodo chiamante (in questo caso dal metodo DammiValore al metodo Main);
riga 9	seconda chiamata del metodo DammiValore: il valore che tale metodo darà in output al Main, dopo l'inserimento da tastiera da parte dell'utente, è assegnato alla variabile b;
riga 10	chiamata del metodo Elaborazione: da notare fra parentesi tonde i nomi delle variabili del Main che contengono i valori che esso deve passare al metodo perché quest'ultimo possa elaborarli; il risultato sarà memorizzato nella variabile s;
riga 32	intestazione del metodo Elaborazione che è stato richiamato dal Main: da notare fra parentesi tonde il nome di due variabili del metodo che vengono inizializzate con i valori della corrispondenti variabili del Main indicate nella chiamata). Si parla di passaggio di parametri per valore (vedi anche la scheda tecnica al termine del capitolo con le definizioni rigorose, ora è importante comprendere il meccanismo di funzionamento!): i valore delle variabili a, b del Main vengono copiati nelle variabili x,y del metodo Elaborazione (equivale alla fotocopia fatta dal responsabile dell'ufficio per tenere al sicuro i documenti originali e consentire comunque all'impiegato di conoscere i dati da elaborare).
riga 34	vengono elaborati i dati delle variabili x,y del metodo, calcolato il risultato e memorizzato nella variabile s del metodo
riga 35	viene restituito al Main il risultato dell'elaborazione
riga 11	il Main chiama subito un altro metodo, passandogli il valore contenuto nella variabile s; Il metodo chiamato dovrà ricevere tale valore e memorizzarlo in una sua propria variabile, in quanto non potrà ovviamente accedere alla variabile del Main;
riga 35	intestazione del metodo Visualizza che è stato richiamato; la variabile risultato di tale metodo viene inizializzata con il valore della corrispondente variabile s del Main; void prima del nome del metodo indica che esso non restituisce alcun valore al Main
riga 40:	visualizza il risultato dell'elaborazione
riga 12:	visualizza il messaggio "-- Grazie per aver utilizzato il nostro software --"
riga 13:	attende la pressione di un tasto da parte dell'utente per chiudere l'applicazione.

Metodo TOP-DOWN
Il problema viene scomposto in sottoproblemi più semplici, ciascuno dei quali risolve uno specifico aspetto del problema complessivo.

5.1.1.4. Parametri e rappresentazione della memoria

Esegui il programma corrispondente al listato proposto e studia le istruzioni man mano che vengono eseguite (con Visual Studio esecuzione con F11-vedi paragrafo 3.2.1.4-, con SharpDeveloper utilizza i breakpoint –vedi 3.2.2.1-), osservando con attenzione la finestra delle variabili locali, che indica di volta in volta le variabili dei vari metodi che si stanno eseguendo con i rispettivi valori e confrontando ciò che vedi con la corrispondente rappresentazione grafica della RAM mostrata in figura.

	Rappresentazione grafica RAM	Finestra variabili locali
1 **Metodo Main** **Riga 7**		
2 **Metodo DammiValore** **Riga 23** Viene allocata la memoria per le variabili locali del metodo DammiValore Le variabili dl Main non sono visibili perché ora è in esecuzione l' altro metodo		
3 **Metodo DammiValore** **-Riga 28**		

	Rappresentazione grafica RAM	Finestra variabili locali
4 **Metodo Main** **Riga 10** Le variabili del Main sono tornate visibili, con i nuovi valori		
5 **Metodo Elaborazione** **Riga 33** La variabili del Main ancora non visibili, i valori sono passati nelle corrispondenti variabili del metodo Elabora		

5.1.1.5. Ambito di una variabile

L' esempio del paragrafo precedente ha mostrato che nei diversi punti del programma risultano visibili, quindi utilizzabili, solo alcune variabili.

Si dice **ambito di una variabile** (**scope** in inglese) l'area in cui la variabile è visibile e si può quindi accedere ad essa: le **variabili locali** sono visibili nell'ambito del blocco[65] in cui esse sono dichiarate.

Due variabili dichiarate con lo stesso nome in metodi diversi sono variabili diverse!
Per le variabili locali inoltre la **durata di una variabile** coincide con il suo scope.

Un ragionamento più completo si potrà fare considerando le variabili di classe.

[65] Un blocco è una sequenza di istruzioni racchiuse fra parentesi graffe { }

5.1.2. Parametri per valore e per riferimento

Presentiamo qui il concetto di parametro, uno dei fondamenti dello sviluppo delle applicazioni complesse.

5.1.2.1. Definizioni

Metodo: blocco di istruzioni associato ad un nome;
Definizione di un metodo: specifica del metodo: si compone della intestazione e del corpo;
Intestazione di un metodo: la prima riga della definizione; specifica il nome del metodo, preceduto dal tipo di valore restituito e seguito dalle parentesi tonde che racchiudono eventuali parametri formali
Corpo di un metodo: insieme di istruzioni racchiuse nel blocco specificato dopo l'intestazione;
Chiamata di un metodo: attivazione del metodo chiamato effettuata da parte del metodo chiamante, che comporta l'esecuzione delle istruzioni del corpo del metodo chiamato: si compone del nome dl metodo seguito dalle parentesi tonde che racchiudono eventuali parametri attuali;
Variabile locale: variabile che appartiene al metodo: un metodo non può accedere alle variabili locali degli altri metodi;
Valore restituito da un metodo: risultato calcolato dal metodo chiamato e fornito al metodo chiamante;
Parametro: variabile locale che consente per la comunicazione, cioè il passaggio di valori, fra metodo chiamante e metodo chiamato: tale passaggio può essere effettuato per valore o per indirizzo(vedi paragrafi successivi); viene inizializzata dal metodo chiamante;
Parametro formale: nome generico di variabile locale del metodo chiamato che rappresenta il valore che verrà fornito dal metodo chiamante;
Parametro attuale: valore fornito dal metodo chiamante e utilizzato dal metodo chiamato.

Tabella che associa le definizioni con gli esempi di codice mostrati nel capitolo precedente

Metodi	Codifica C# 5-1	Figura 5-2	Codifica C# 5-2
Nome del Metodo	Main, Presentazione, Elaborazione, Visualizza	Main, M1	Main, Tempo

Definizione di un metodo	Presentazione: righe 17-20
	DammiValore: righe 22-30
	Elaborazione: righe 32-36
	Visualizza: righe 38-41
		static void M1 (int x, int y)
		{
		// corpo del metodo
		}
			Main: righe 4-16
			Tempo: righe 18-23

Intestazione di un metodo	Presentazione: riga 17 DammiValore: riga 22 Elaborazione: riga 32 Visualizza: riga 38	static void M1 (int x, int y)	Main: riga 4 Tempo: riga 18
Corpo di un metodo	Presentazione: riga 19 DammiValore: righe 24-29 Elaborazione: righe 34-35 Visualizza: riga 40		Main: righe 6-15 Tempo: righe 20-22
Chiamata di un metodo	Righe 7, 8, 9, 10 e 11	M1(a,b);	Righe 9, 11, 13

Variabili	Codifica C# 5-3	Figura 5-3	Codifica C# 5-4
Variabili locali	Metodo Presentazione: nessuna Metodo DammiValore: s, tmp Metodo: x,y,s	metodo Main: a,b metodo M1; x,y	Main: oreA, minutiA, oreB, minutiB,risultato Metodo Tempo: ore, minuti, totale
Valore restituito	Presentazione: nessuno Riga 29 - DammiValore: s Riga 35 - Elaborazione: s Visualizza: nessuno	metodo M1:nessuno	Metodo Tempo: Totale
Parametri formali	Riga17- Presentazione: nessuno Riga22- DammiValore: nessuno Riga32 - Elaborazione: x, y Riga 38 - Visualizza: risultato	metodo M1: x,y	Metodo Tempo: ore, minuti
Parametri attuali	Riga 7- Presentazione: nessuno Riga 8,9:DammiValore:nessuno Riga10 - Elaborazione: a,b Riga 11 - Visualizza: s	metodo Main: a,b	Riga 9: oreA, minutiA Riga 11: oreB, minutiB Riga 13: 2,5

5.1.2.2. Passaggio di parametri per valore

Quando un parametro viene passato per valore il metodo chiamante passa al metodo chiamato i valori delle variabili; in memoria si ha una copia di questi dati, per cui il metodo chiamato non può accedere ai valori originali, e quindi non li può modificare (questa affermazione è valida in senso stretto solo per variabili di tipo semplice, ad esempio interi, double, ma non per array e oggetti in genere). Nota bene: vi è una corrispondenza posizionale fra parametri attuali e parametri formali; la variabile del metodo chiamato viene inizializzata con il valore della corrispondente variabile del metodo chiamante.

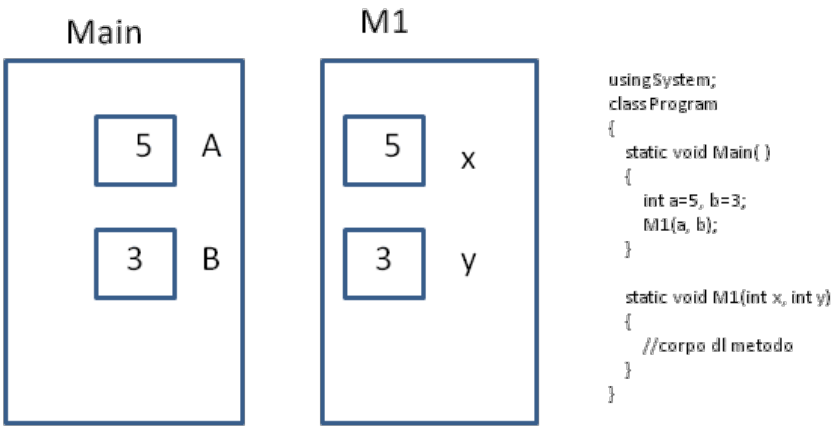


Figura 5-4 - parametri: passaggio per valore

.

5.1.2.3. Restituzione di un valore

Si conosce il tempo, espresso in ore e minuti, impiegato da un atleta per compiere un determinato percorso: si vuole calcolare il tempo in minuti.

Analisi

Proviamo a risolvere il problema con il metodo top-down pensando che in generale l'elaborazione richiesta dovrà essere effettuata sui dati di più atleti. Risulta quindi spontaneo individuare il seguente sottoproblema: per calcolare il tempo impiegato da un generico atleta bisogna conoscere i valori precisi delle ore e dei minuti di quello specifico atleta (in termini tecnici: i valori attuali), ma per progettare la soluzione del problema possiamo fare riferimento a due generiche variabili. Effettuato il calcolo dovremo restituire, cioè passare, comunicare, il risultato al richiedente[66].

Possiamo rappresentare in forma grafica il nostro ragionamento, indicando i nomi che scegliamo per le variabili corrispondenti ai valori che il metodo Tempo deve ricevere in Input dal metodo chiamante ed al valore che deve restituire in output come risultato.

Proviamo ora documentare il nostro lavoro descrivendo i metodi secondo le specifiche introdotte.

Nome del metodo: Main

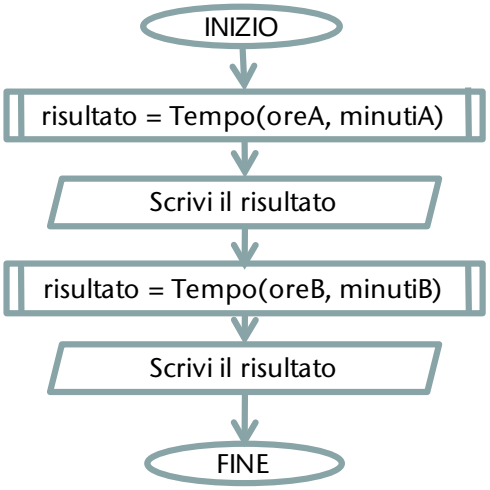
Scopo: conserva i dati e coordina il lavoro, chiamando i diversi metodi ogni volta che c'è da svolgere una attività specifica

Tabella di descrizione delle variabili locali del metodo Main

Nome	Descrizione	Tipo di dato
oreA	Ore impiegate dall'atleta A	numero intero
oreB	Ore impiegate dall'atleta B	numero intero
minutiA	minuti impiegati dall'atleta A	numero intero
minutiB	minuti impiegati dall'atleta B	numero intero
risultato	Tempo totale impiegato da un certo atleta espresso in minuti	numero intero

[66] A livello di codifica seguirà che nel Main si associa a tale elaborazione l' identificatore di un metodo, ad esempio Tempo; con la chiamata del metodo verrà automaticamente effettuato il calcolo che ci interessa. Ogni volta che il computer *automaticamente* svolge un compito c'è naturalmente il lavoro di chi ha specificato le istruzioni da svolgere (la definizione del metodo).

Algoritmo Main



Nome del metodo: Tempo

Scopo: ricevere i dati relativi al numero di ore e di minti impiegati da un atleta calcolare il tempo totale espresso in minuti

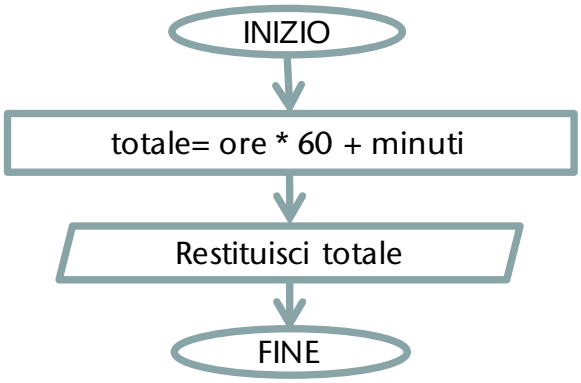
Dati di input(cioè dati ricevuti dal metodo):
dal metodo chiamante:
da tastiera:

Dati di output (cioè dati forniti, o restituiti, dal metodo):
al metodo chiamante:
A video:

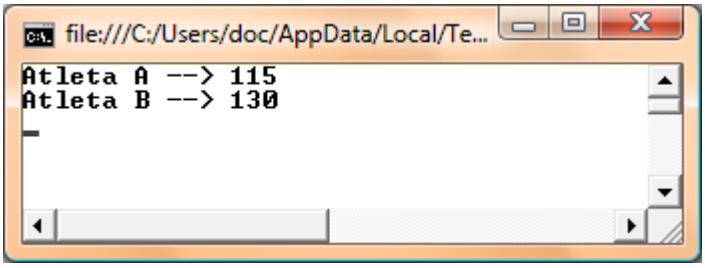
Tabella di descrizione delle variabili locali del metodo Tempo

Nome	Descrizione	Tipo di dato	Input/Output/lavoro
ore	Ore impiegate da un atleta	numero intero	Input dal metodo chiamante
minuti	Minuti impiegati da un atleta	numero intero	Input dal metodo chiamante
totale	Tempo totale in minuti	numero intero	Output al metodo chiamante

Algoritmo Tempo



Osserva e leggi il codice corrispondente




```
1   using System;
2     class Atleta
3     {
4        static void Main(string[] args)
5        {
6           int oreA = 1, minutiA = 55; //ore e minuti dell' atleta A;
7           int oreB = 2, minutiB = 10; //ore e minuti dell' atleta B;
8           int risultato;
9     (3)   risultato = Tempo(oreA, minutiA);
10          Console.WriteLine("Atleta A --> {0}", risultato);
11          risultato = Tempo(oreB, minutiB);
12          Console.WriteLine("Atleta B --> {0}", risultato);
13          Console.ReadLine();
14       }
15
16       static int Tempo(int ore, int minuti)
17       {
18          int totale;
19          totale = ore * 60 + minuti;
20    (5)   return totale;
21       }
22    }
```

(1) (1) (4) (2) (2)

sequenza di esecuzione delle istruzioni:

Metodo Main	6, 7, 8,9
Metodo Tempo	18,19,20
Metodo Main	10, 11
Metodo Tempo	18,19,20
Metodo Main	12,13

Codifica C# 5-5 - metodi: restituzione di un valore

Analizziamo in dettaglio il listato, indicando le istruzioni che vengono eseguite

1	oreA, minutiA oreB, minutiB 2,15	Parametri attuali, passaggio per valore: vengono cioè passati al metodo chiamato i valori specificati nelle varie chiamate del metodo stesso
2	Ore, minuti	Parametri formali, specificati nella definizione del metodo, corrispondono ai valori in input dal metodo chiamante
3	risultato	Variabile locale del Main che riceve il valore restituito dal metodo chiamato
4	Int	Tipo di valore restituito dal return
5	return	istruzione che termina l'esecuzione del metodo restituendo il risultato al metodo chiamante: il valore calcolato e restituito deve essere dello stesso tipo indicato nella intestazione (riga 18)

Ultima considerazione: ricorda che possiamo avere un metodo che non deve restituire valori al metodo chiamante, ad esempio perché il suo scopo è semplicemente quello di produrre un output a video: l'esecuzione del metodo è comunque terminata dall'istruzione return, che in questo caso non è seguita da alcun nome di variabile[67] e nell'intestazione, come tipo del dato restituito da indicare prima del nome del metodo, si specifica void .

Ricorda

static tipoValoreRestituito **NomeMetodo (** listaParametriFormali **)** **{** //variabili locali i cui valori non sono condivisi con altri metodi //istruzioni del metodo **}**	**definizione del metodo**
static tipoValoreRestituito NomeMetodo **(tipo1 id1, tipo2 id2)** e così via per ogni parametro; id1, id2 sono i nomi delle variabili	**listaParametriFormali**
static **tipoValoreRestituito** NomeMetodo **(** listaParametriFormali **)** il tipo del valore restituito può essere int, double, e così via	**tipoValoreRestituito** void se non restituisce nessun valore
static tipoValoreRestituito NomeMetodo **(** listaParametriFormali **)** **{** //variabili locali i cui valori non sono condivisi con altri metodi //istruzioni del metodo **return risultato;** **}**	**return** chiude il metodo, eventualmente restituendo un risultato

[67] Return può essere omesso, in questo caso il metodo termina dopo l' esecuzione dell' ultima istruzione

Consideriamo un altro esempio

scrivere un metodo che, ricevendo dal metodo chiamante due valori, restituisca il maggiore

Codifica

```
using System;
class Program
{   static void Main( )
    {
       int a = 6, b = 7;
       Console.WriteLine(Massimo(a, b));
       Console.WriteLine("\nPremi INVIO per chiudere il programma");
       Console.ReadLine();
    }

    static nt Massimo(int x, int y)
    {
       if (x > y)
       {
          return x;
       }
       else
       {
          return y;
       }
    }
}
```

Osserva : return chiude il metodo , il ramo else non viene eseguito se il primo valore è maggiore del secondo

5.1.2.4. Passaggio di parametri per riferimento

Quando un parametro viene passato per riferimento il metodo chiamante concede al metodo chiamato l'accesso all'area in cui sono memorizzati i valori delle proprie variabili, quindi il metodo chiamato può modificare tali valori. Non viene creata una copia dei dati, i parametri del metodo chiamato accedono all'area di memoria della variabili del metodo chiamante mediante riferimenti (puntatori) alle stesse.

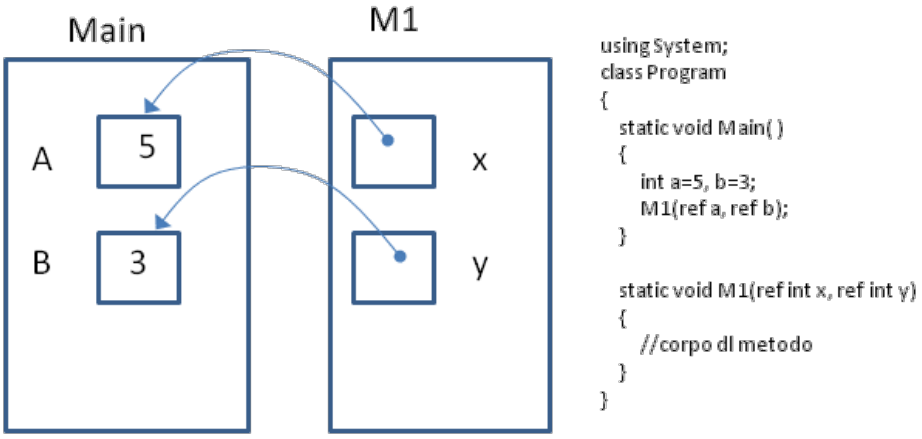


Figura 5-5 - parametri: passaggio per riferimento

Scrivere un metodo che, ricevuti due valori interi, li restituisca in ordine crescente.

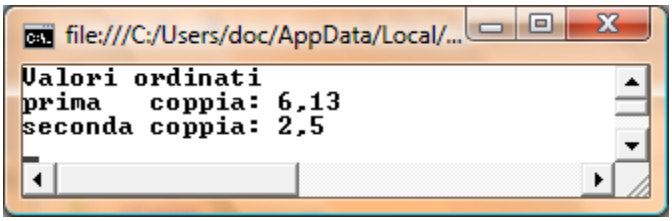
Analisi
Per ordinare i due valori è sufficiente confrontarli fra loro: se il secondo è maggiore del primo sarà necessario scambiarli.
Rispetto alla logica il problema è semplice, qualche considerazione sui dati è però opportuna.
Il metodo deve ricevere due valori interi, quindi si possono prevedere due parametri. Come risultato dobbiamo restituire i due valori ordinati: utilizzando return è possibile restituire un solo valore! Sarà allora necessario che il metodo possa eventualmente modificare direttamente i valori memorizzati dal metodo chiamante, ciò che si può realizzare con un **passaggio di parametri per riferimento**.

Nome del metodo: Scambia
Scopo: ricevere due valori interi e li restituisce scambiati
Dati di input(cioè dati ricevuti dal metodo):
dal metodo chiamante: due valori interi
da tastiera: nessuno
Dati di output (cioè dati forniti, o restituiti, dal metodo):
al metodo chiamante: due valori interi
A video: nessuno

Tabella di descrizione delle variabili locali del metodo Scambia

Nome	Descrizione	Tipo di dato	Input/Output/lavoro
a	primo valore ricevuto, in output valore minore	numero intero	Input dal metodo chiamante e output al metodo chiamante
b	secondo valore ricevuto, in output valore minore	numero intero	Input dal metodo chiamante e output al metodo chiamante
x	variabile di appoggio per lo scambio	numero intero	lavoro

Osserva e leggi il codice corrispondente[68]



[68] La variabile x del metodo Main non ha nient e a che fare con la variabile x el metodo Scambia: attenzione a non farsi confondere dai nomi!

```
using System;
 class Program
   { static void Main( )
      {
         int x=13, y=6, w=2, z=5;
      1  Scambia(ref x, ref y);
         Scambia(ref w, ref z);
         Console.WriteLine("Valori ordinati");
         Console.WriteLine("prima   coppia: {0},{1}",x,y);
         Console.WriteLine("seconda coppia: {0},{1}",w,z);
         Console.ReadLine();
      }
      3      2      2
      static void Scambia(ref int a, ref int b)
      {
         if (a > b)
         {
            int x;
            x = a;
            a = b;
            b = x;
         }
      }
   }
```

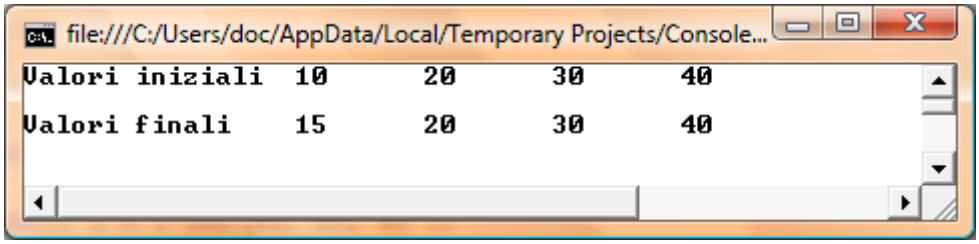
Codifica C# 5-6- parametri: passaggio per riferimento

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

1	ref x, ref y	chiamata del metodo con parametri attuali: il metodo chiamato riceve i riferimenti a queste variabili, quindi potrà conoscere i valori memorizzati ed anche modificarli
2	ref int a, ref int b	intestazione del metodo con parametri formali passati per riferimento (ref)
3	void	non c'è valore di ritorno (non si utilizza return con nome variabile)

5.1.2.5. Variabili di tipo riferimento come parametri

Osserva e leggi il codice corrispondente




```
using System;
   class Program
   {
      static void Main( )
      {
      1 int[] V = { 10, 20, 30, 40 };
        Console.Write("Valori iniziali\t");
        for (int i = 0; i < 4; i++)
           Console.Write(" {0}\t", V[i]);
      2 Modifica(V);    //passaggio di parametro per valore
                        // variabile di tipo riferimento
                        // il metodo Modifica i valori memorizzati nel vettore
        Console.Write("\n\nValori finali\t");
        for (int i=0; i<4; i++)
      4   Console.Write(" {0}\t",V[i]);
        Console.ReadLine();
      }
      static void Modifica(int[] Vettore)
      {
      3 Vettore[0] = 15;
      }
   }
```

1. dichiarazione ed inizializzazione del vettore V, variabile del metodo Main. Questi valori iniziali vengono subito visualizzati (codice righe 7-9)

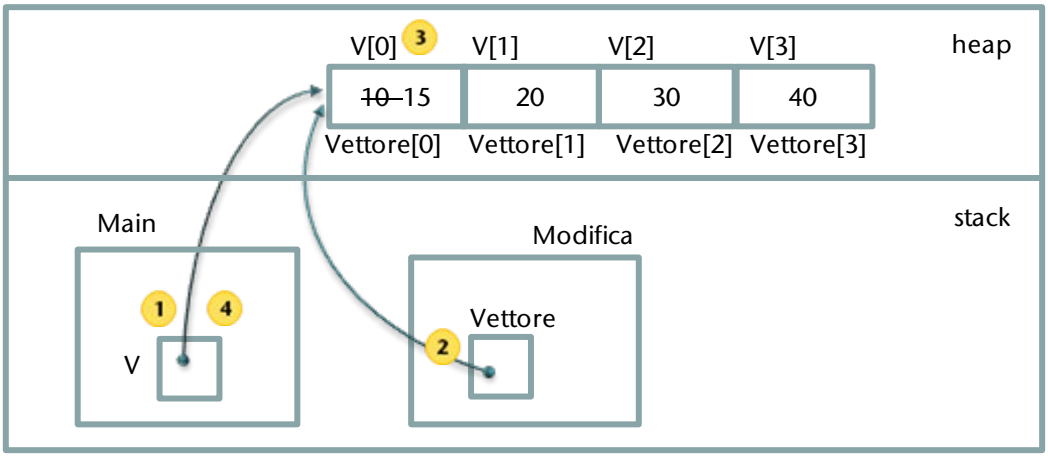
2. Viene chiamato il metodo Modifica, a cui viene passato il vettore V, che è una variabile di tipo riferimento (vedi paragrafo 0). Fai attenzione: non vengono passati i singoli valori memorizzati nei diversi elementi del vettore, ma il riferimento all'area di memoria in cui essi sono memorizzati. Siccome abbiamo un passaggio per valore il metodo Modifica non potrà modificare il valore di V, ma nessuna garanzia si può avere sui singoli valori memorizzati.

3. Il metodo Modifica agisce sul primo elemento del nostro array. A quale area di memoria fa riferimento la variabile Vettore? Vettore è un parametro passato per valore, quindi viene inizializzato dal Main con la chiamata del metodo: in questo caso assume il valore di V, cioè fa riferimento agli stessi elementi cui fa riferimento V.

4. Siamo tornati al Main, V fa riferimento alla locazione i memoria in cui si trova il primo elemento del vettore, proprio la stessa area di memoria che era stata modificata dal metodo!

Per comprendere bene come questo possa avvenire riflettiamo ancora su quanto avviene nella memoria RAM: quando la variabile è di tipo riferimento non vengono duplicati i valori memorizzati nella struttura, ma viene copiato solo il valore del riferimento (puntatore).

Nel nostro esempio si ha una copia della variabile V, cui corrisponde la variabile Vettore del metodo Modifica che, avendo lo stesso valore di V, farà riferimento agli stessi dati.
Di conseguenza per ogni locazione di memoria cui corrisponde un dato elemento del vettore si hanno due nomi distinti, utilizzati dai due diversi metodi (chiamante e chiamato). In particolare quando il metodo Modifica aggiorna il valore di Vettore**[0]** assegnando il valore 15 (e sovrascrivendo il valore **10** memorizzato in precedenza), va a scrivere nella stessa locazione di memoria alla quale in seguito farà accesso il metodo Main, chiamandola V**[0].**



Questa stessa situazione si verifica con ogni variabile di tipo riferimento: quando si ha un passaggio di parametri per valore il metodo chiamato può modificare i dati della variabile del metodo chiamante.

5.1.3. Ricorsione

La ricorsione è un metodo molto importtante per la soluzione dei problemi, alternativo all'approccio iterativo. Viene qui presentato un esempio di calcolo ricorsivo, mettendo in evidenza a un lato la semplicità e l'eleganza del ragionamento, dall'altra l'utilizzo delle risorse hardware, nello specifico della memoria RAM, richieste per l'esecuzione di una applicazione ricorsiva..

5.1.3.1. Calcolo del fattoriale di un numero intero

Ricordiamo la definizione matematica di *fattoriale di un numero n* intero positivo:

$$n! = 1 * 2 * 3 * \ldots * (n - 1) * n$$

È possibile dare anche l'equivalente definizione ricorsiva, in cui un concetto viene definito mediante sé stesso:

$$\begin{cases} n! = 1 & n=1 \\ n! = n\,(n-1)\,! & n>1 \end{cases}$$

Osserva e leggi il codice corrispondente, sviluppato nelle due versioni

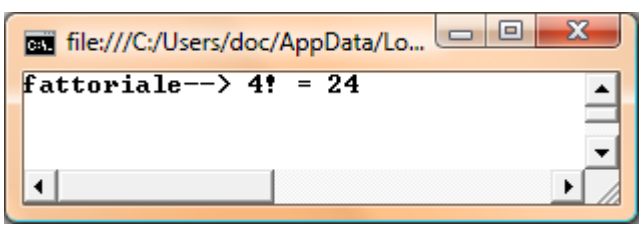
Soluzione iterativa

```
1  using System;
2  class Program
3  {
4   static void Main(string[] args)
5   {
6     int n = 4, y = 1;
7     for (int i = 1; i <= n; i++)
8        y = y * i;
9     Console.WriteLine("fattoriale--> 4! = " + y);
10    Console.ReadLine();
11   }
12 }
```

Soluzione ricorsiva

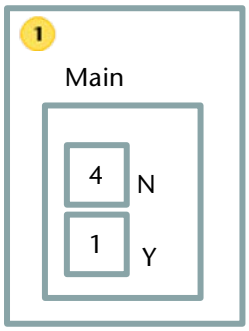
```
1  using System;
2   class Program
3  {
4     static void Main(string[] args)
5     {
6        int n=4, y=1;
7        y = fatt(n);
8        Console.WriteLine("fattoriale--> 4! = " + y);
9        Console.ReadLine();
10    }
11
12    static int fatt (int V)
13    {
14       int R; //risultato
15       if (V == 1)
16          R=1;
17       else
18          R = V * fatt(V - 1);
19       return R;
20    }
21 }
```

Le due versioni sono equivalenti per quanto riguarda il risultato prodotto (è diversa la complessità della soluzione, riguardo al tempo di esecuzione ed allo spazio di memoria allocato).

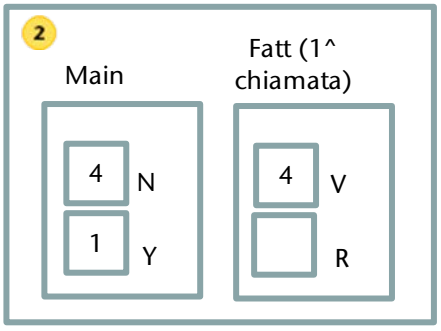


Analizziamo in dettaglio il listato della soluzione ricorsiva, indicando man mano le istruzioni che vengono eseguite ed esaminando quanto avviene in memoria

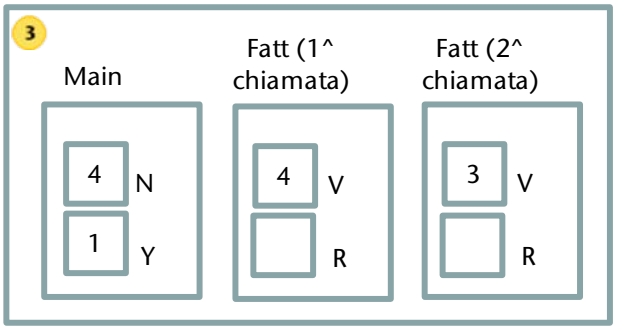
1 riga 6 vengono inizializzate le variabili del Main;



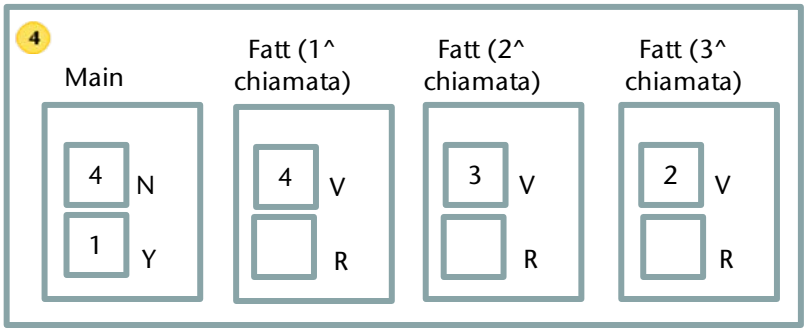
2 riga 7 e 12 chiamata del metodo fatt (il valore 4 della variabile N è utilizzato per inizializzare il parametro V)



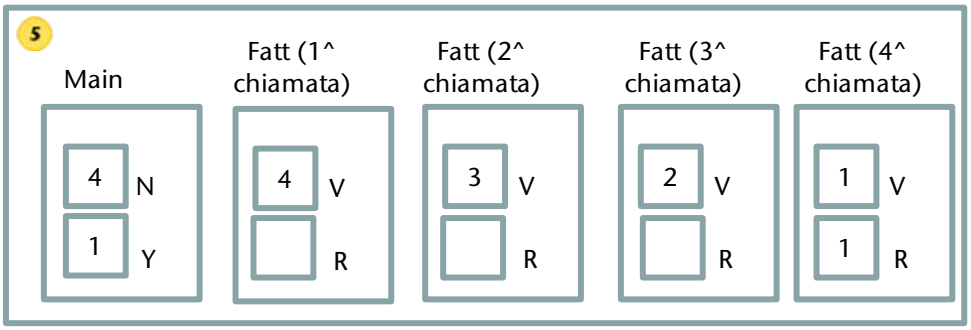
3 riga 15 e18 andiamo nel ramo else e abbiamo fatt(v – 1) →viene cioè effettuata una nuova chiamata del metodo fatt (con valore attuale (v-1) che equivale a (4-1) cioè a 3; il risultato ottenuto verrà moltiplicato per V (cha ha valore 4) e verrà assegnato alla variabile R. Per ora siccome c'è la nuova chiamata, si modifica la memoria e viene passato il valore 3;



4 riga 15 e18 andiamo ancora nel ramo else e abbiamo fatt(v – 1) → nuova chiamata del metodo fatt (con valore attuale (v-1) che equivale a (3-1) cioè a 2; si modifica ancora la memoria;



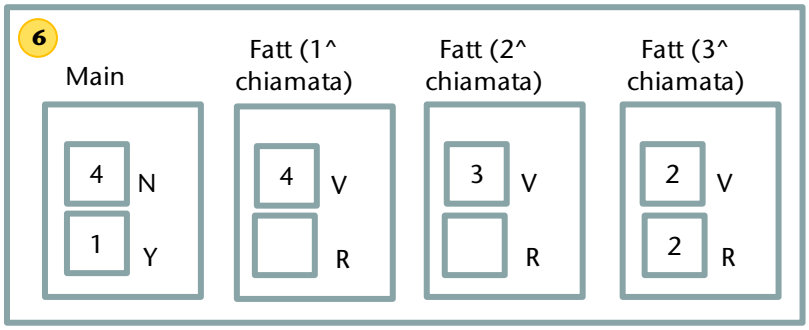
5 riga 15 e 18



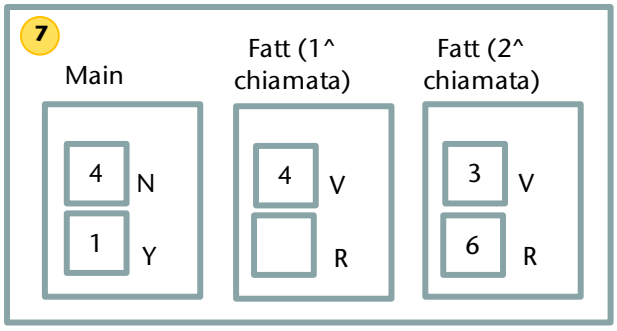
come sopra, andiamo ancora nel ramo else e abbiamo fatt(v – 1) → nuova chiamata del metodo fatt (con valore attuale (v-1) che equivale a (2-1) cioè a 1; si modifica ancora la memoria;

5 riga 15 e16 ora V=1, quindi si esegue l'istruzione R=1;

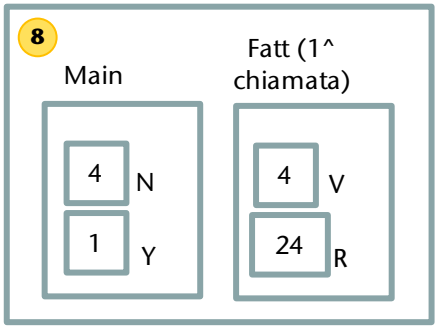
6 riga 19 viene restituito il valore di R (cioè 1) al metodo chiamante e si chiude l'esecuzione di fatt (4^ chiamata), il controllo ritorna a fatt (3^ chiamata) il quale ora può completare l'esecuzione della istruzione 18 ed aggiornare il proprio valore di R; ricordiamo R=V*fatt(v-1) dove fatt(v-1) è il valore dalla variabile R restituito dalla chiamata 4.



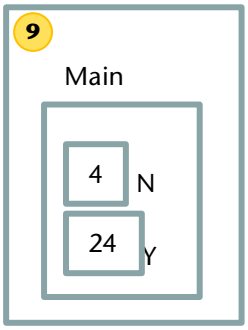
7 Come sopra



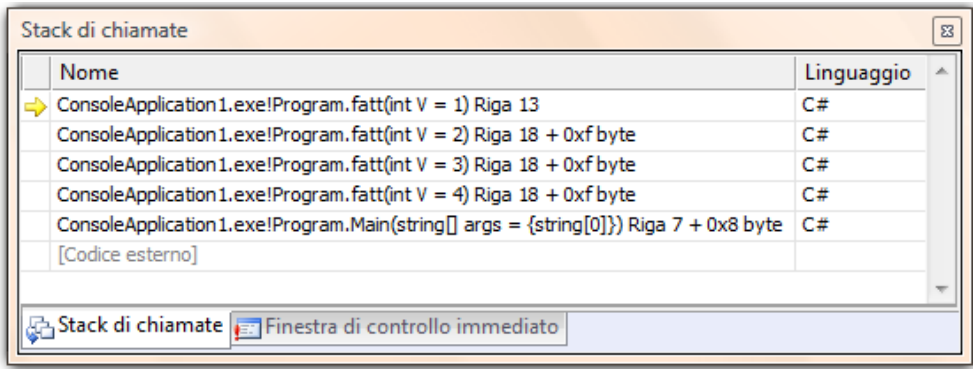
8 Come sopra



9 riga 7
Viene restituito al Main il risultato finale



Osserva l'esecuzione di questo codice mostrata da Visual Studio, con tutte le chiamate ricorsive al metodo fatt.



5.2. SVILUPPO DI APPLICAZIONI

PREREQUISITI

Saper risolvere problemi -

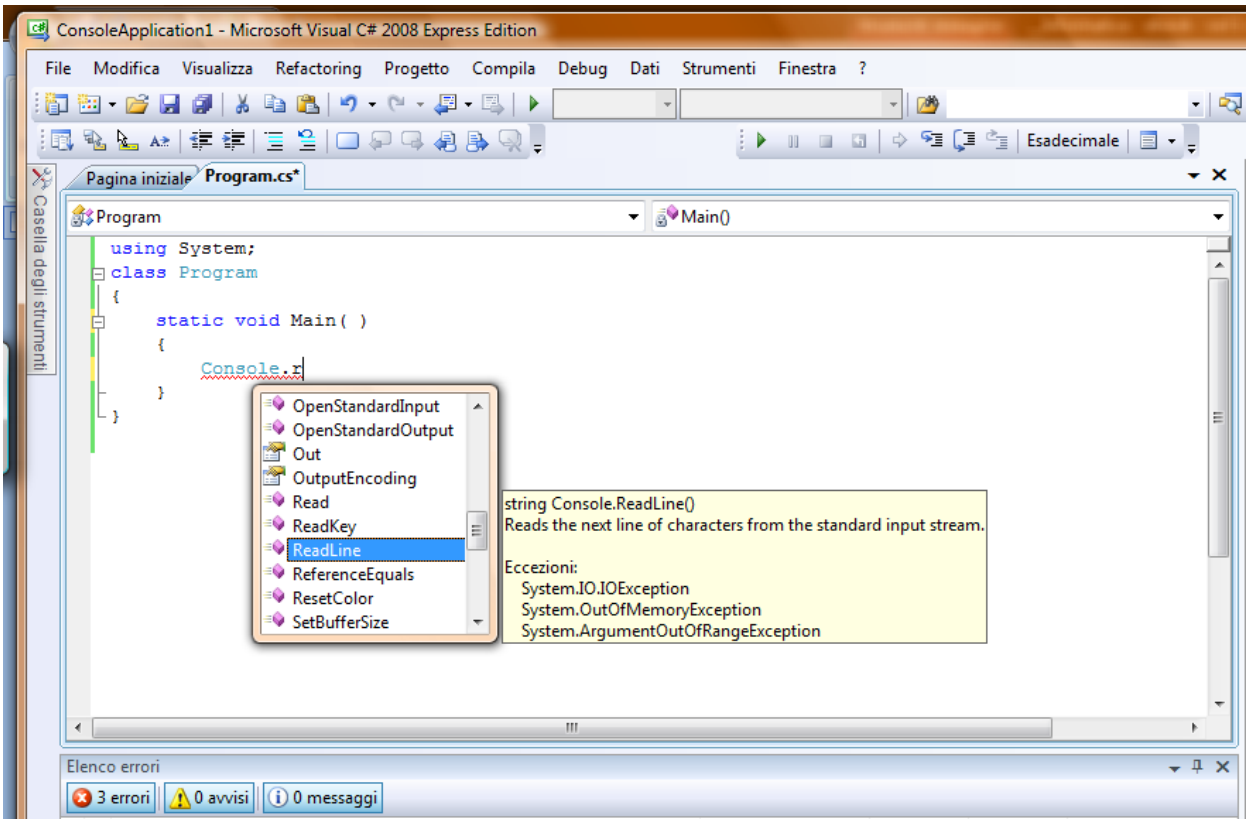
OBIETTIVI

Sapere sviluppare applicazioni

5.2.1. Utilizzo di metodi del .NET Framework

Impariamo ad utilizzare i metodi del .NET Framework ed a sviluppare applicazioni con l'uso di metodi definiti anche da noi stessi.

5.2.1.1. Gestione Input/output



Osserviamo questa videata e riflettiamo :

- quando si scrive Console seguita da un punto interviene intelliSense (vedi paragrafo 3.2.1.3) a mostrare, identificandoli con una icona che rappresenta un piccolo box, tutti i metodi previsti per essa
- selezionando un metodo in particolare viene mostrata la sintassi per utilizzarlo: nel caso specifico vediamo lo scopo del metodo e osserviamo che non prevede parametri, ma che invece restituisce un valore di tipo stringa. Ora dovrebbe essere del tutto chiaro il significato delle istruzioni seguenti:

  string tmp; tmp = Console.ReadLine();

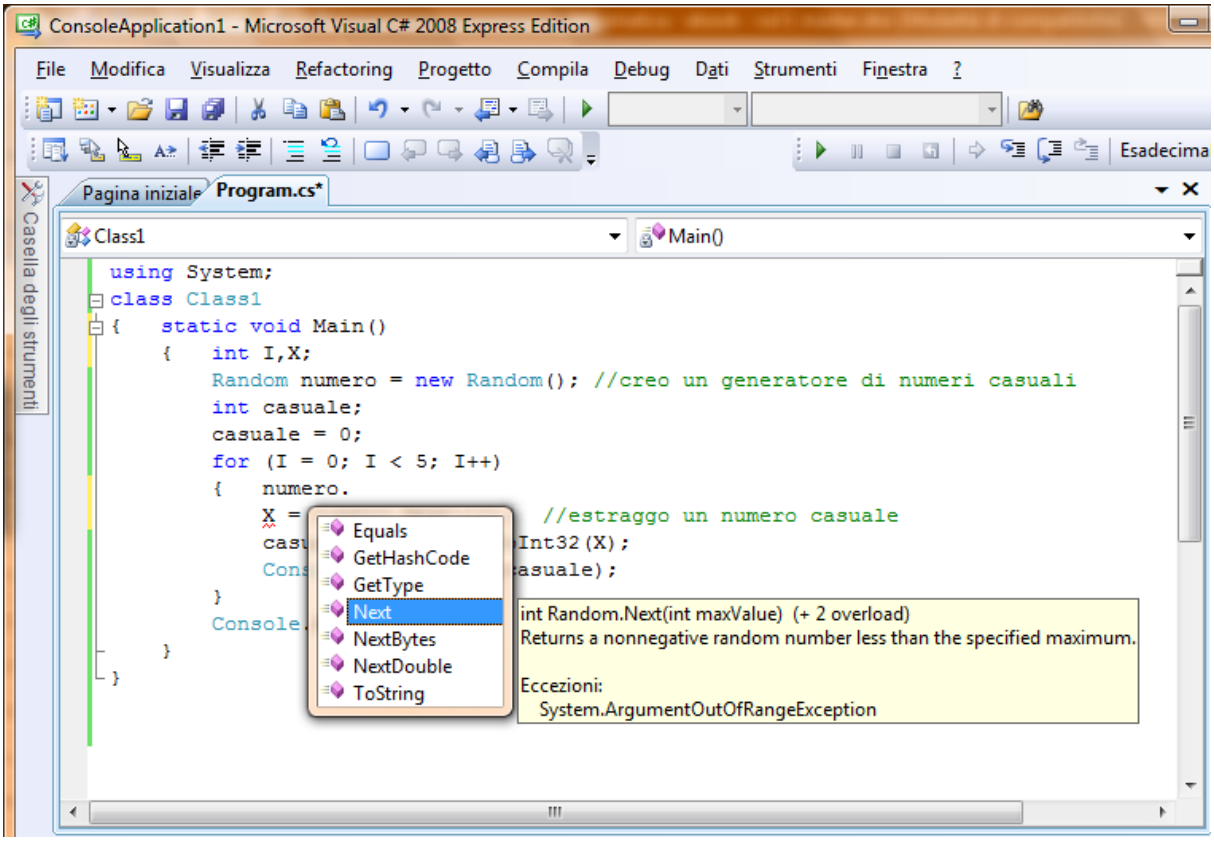
5.2.1.2. Generazione numeri casuali

Problema→ ottenere 5 numeri casuali compresi fra zero e dieci.

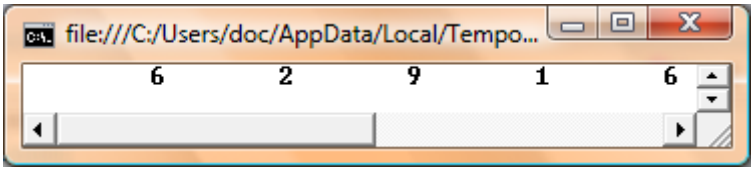
Analisi generale

Cosa posso fare per risolvere il problema? Ricordiamo l'esempio proposto al paragrafo 5.1.1.1 relativo all'impresa edile. Un modo eseguire dei lavori è vedere se troviamo dei componenti prefabbricati che ci possano essere utili.

Stesso procedimento utilizziamo noi: c'è un metodo per estrarre i numeri casuali? L' esperienza e lo studio ci possono guidare: tale metodo esiste, è sufficiente utilizzarlo nella nostra applicazione. Ancora: per utilizzarlo è sufficiente richiamarlo e definire in modo opportuno tutti i parametri, lasciandoci guidare al solito da IntelliSense, come mostrato in figura.



Osserva e leggi il codice corrispondente




```
using System;
class Class1
{  static void Main()
   {  int I,X;
      Random numero = new Random(); //creo un generatore di numeri casuali
      int casuale;
      casuale = 0;
      for (I = 0; I < 5; I++)
      {   X = numero.Next(10);  //estraggo un numero casuale
         casuale = Convert.ToInt32(X);
         Console.Write("\t"+casuale);
      }
      Console.ReadLine();
   }
}
```

5.2.2. Sviluppo di applicazioni con l' uso dei metodi

Si presenta qui un caso di sviluppo di applicazioni con l' uso dei metodi.

5.2.2.1. Gestione alunni (soluzione con utilizzo di metodi e parametri)

Problema→ si ripropone il problema già presentato in precedenza relativo alla gestione di un elenco di alunni con i relativi voti (vedi paragrafo 4.3.2.2) , ma si richiede ora una soluzione che utilizzi il metodo top-down.

Analisi generale

Il problema è analogo al precedente per la gestione delle diverse funzionalità (inserimento dei dati, visualizzazione tabella con nomi e voti, selezione e conteggio alunni bravi) e può essere gestito come si è visto presentando all' utente un menu per la scelta fra le diverse funzionalità dell' applicazione.

Osserviamo ora che a ciascuna voce del menu corrisponde un preciso piccolo problema, che si configura come sottoproblema del problema principale: ciascuno di questi sottoproblemi verrà gestito in modo autonomo , tenendo conto comunque che ognuno di essi opera su dati comuni al problema generale.

La soluzione del problema generale si basa quindi semplicemente sulla presentazione all'utente del menu e sulla gestione della scelta che si effettua attraverso la chiamata del metodo corrispondente.
La memorizzazione dei dati dell' utente sarà compito del metodo Main, che dovrà anche chiamare gli altri metodo per attivare le diverse funzioni.

Tabella di descrizione delle variabili[69] per la memorizzazione dei dati dell'utente

Nome	Descrizione	Tipo di dato
V[5]	Vettore dei voti	intero
Nomi[5]	Vettore dei nominativi	stringa
sc	Scelta opzione del menu	Numero reale

Indichiamo solo le variabili per la memorizzazione di dati dell' utente (in questo esempio elenco nominativi e voti) e per la gestione del menu in quanto tutte le altre funzionalità verranno prese in carico dagli altri moduli che, per svolgere i propri compiti, avranno a disposizione i dati inseriti dall'utente e memorizzati dal Main.

Algoritmo generale

L'algoritmo che presenta e gestisce il menu è quello già presentato in Figura 4-5 quando in modo intuitivo si erano utilizzati i metodi intesi come sottoprogrammi, senza poter poi scrivere la codifica corrispondente in quanto non si possedeva ancora la tecnica necessaria, in particolare le competenze nell' uso dei parametri.

[69] Non si utilizzano i record in modo che essi non siano un prerequisito per la comprensione di questa parte

Di seguito si analizza, a titolo di esempio, uno dei sottoproblemi. La soluzione completa prevede naturalmente l'analisi di ciascuno dei singoli sottoproblemi che corrispondono, in questo esempio, alle diverse voci del menu.

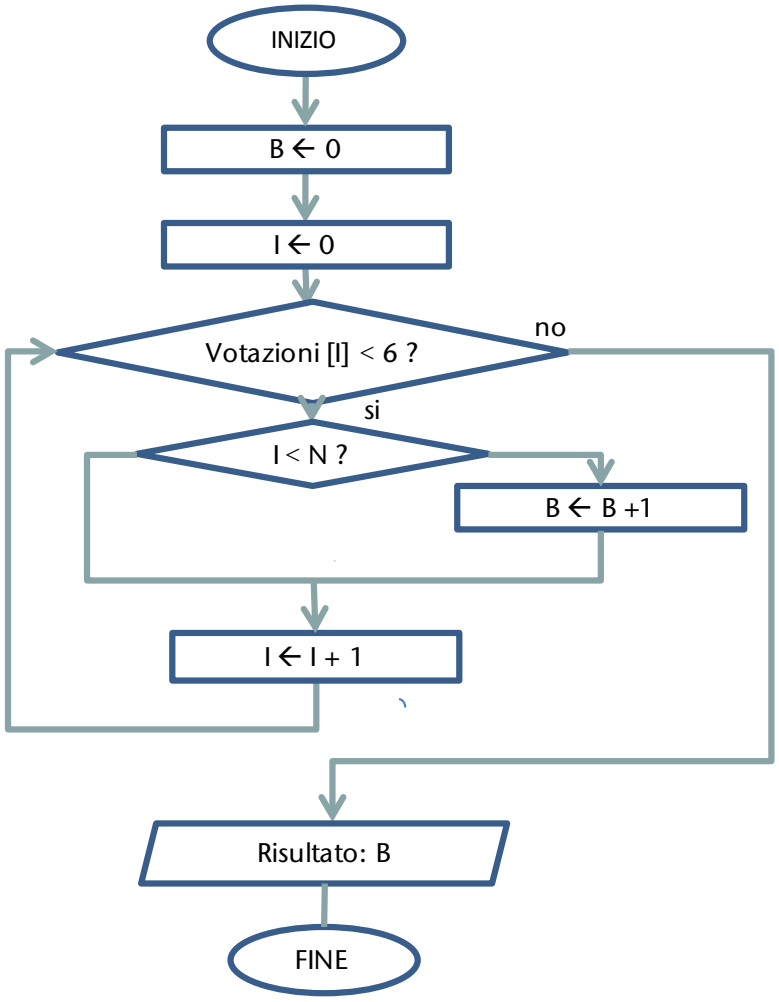
Analisi metodo Bravi
Si esaminano ad uno ad uno tutti i voti e si conteggiano i valori sufficienti mostrando il risultato finale.

Tabella di descrizione delle variabili ConteggioBravi

Nome	Descrizione	Tipo di dato	Input/Output/Lavoro
Votazioni[5]	Vettore dei voti	intero	Input dal Main
B	Contatore numero studenti Bravi, con voto almeno sufficiente	intero	output
i	Contatore, indica la posizione dell' elemento che si esamina	intero	lavoro

Osservare che si sono indicati esplicitamente i dati che vengono passati dal Main per l' elaborazione.

Algoritmo Bravi



Codifica

```
using System;
class Program
{
 static void Main(string[] args)
  {
    int[] V = new int[5];
    string[] Nomi = new string[5];
    int sc;    //scelta opzione menu
    string tmp;
    do
    {
       // presentazione menu
       Console.WriteLine("1. Inserimento nominativi alunni");
       Console.WriteLine("2. Inserimento voti alunni");
       Console.WriteLine("3. Visualizza dati");
       Console.WriteLine("4. Conta alunni bravi");
       Console.WriteLine("5. Elenco alunni recupero");
       Console.WriteLine("0. Esci");
       Console.WriteLine("Indica la tua scelta");
       tmp = Console.ReadLine();
       sc = Convert.ToInt32(tmp);
       switch (sc)
       {
          case 1: InserimentoNomi(Nomi);     break;
          case 2: InserimentoVoti(Nomi, V);  break;
          case 3: Visualizza(Nomi, V);       break;
          case 4: Bravi(V);                  break;
          case 5: Recupero(Nomi, V);         break;
       } //fine switch
    } while (sc != 0);

    Console.WriteLine("\nPremi INVIO per chiudere il programma");
    Console.ReadLine();
  } //fine metodo Main


static void InserimentoNomi(string[] Nominativi)
  {  int i;
     Console.WriteLine("Inserimento nominativi alunni");
     for (i = 0; i < 5; i++)
     {
        Console.WriteLine("dammi il nominativo dell' alunno numero {0}: ", i + 1);
        Nominativi[i] = Console.ReadLine();
     }
  } //fine metodo InserimentoNomi

  static void InserimentoVoti(string []Nominativi, int[] Votazioni)
  {  int i;
     string tmp;
     Console.WriteLine("\nInserimento voti alunni");
     for (i = 0; i < 5; i++)
     {
        Console.WriteLine("dammi il voto dell' alunno  {0}: ", Nominativi[i]);
        tmp = Console.ReadLine();
        Votazioni[i] = Convert.ToInt32(tmp);
     }
  }//fine metodo InserimentoVoti
```

```
    static void Visualizza(string[] Nominativi, int[] Votazioni)
    {  int i;
       Console.WriteLine("\nTabella dati alunni");
        Console.WriteLine("nomi\tvoti");
        for (i = 0; i < 5; i++)
        {
            Console.WriteLine(Nominativi[i] + "\t" + Votazioni[i]);
        }
    } //fine metodo Visualizza


    static void Bravi(int[] Votazioni)
    {   int i, B=0;
        for (i = 0; i < 5; i++)
        {
            if (Votazioni[i] >= 6)
                B = B + 1;
        }
        Console.WriteLine("\nTotale alunni bravi {0}: ", B);
    } //fine metodo Bravi


    static void Recupero(string[] Nominativi, int[] Votazioni)
    {   int i;
        //elenco alunni recupero
        Console.WriteLine("\nElenco alunni recupero");
        for (i = 0; i < 5; i++)
        {
            if (Votazioni[i] < 6)
                Console.WriteLine(Nominativi[i]);
        }
    } //fine metodo Recupero

} //fine classe
```

HAI IMPARATO A

1. Utilizzare la metodologia top-down
2. Scrivere programmi con metodi e parametri

6. Applicazioni Windows

Applicazioni Windows: si introduce qui lo sviluppo di applicazioni con interfaccia grafica a in cui l'interazione con l'utente è guidata dagli eventi.

Si ritiene essenziale comprendere i meccanismi di base, quindi si propone di cominciare scrivendo tutto il codice necessario, di seguito si impara ad utilizzare Visual Studio come ambiente per la programmazione visuale, andando comunque a leggere e a comprendere il codice generato in forma automatica.

Si presentano i controlli di base di una interfaccia grafica: Form, Label, TextBox, Button,MessageBox.

- Sviluppo di interfacce grafiche
- Programmazione visuale

6.1. SVILUPPO DI INTERFACCE GRAFICHE

PREREQUISITI

[conoscere gli elementi di base della programmazione in linguaggio C#]

OBIETTIVI

[Saper scrivere la codifica di semplici applicazioni windows e saper utilizzare ambienti di sviluppo visuale]

6.1.1. Windows Form

6.1.1.1. La prima applicazione Windows: oggetto Form

Osserva e leggi il codice corrispondente

```
1  using System;
2  using System.Windows.Forms;
3
4  class Form1: Form
5  {
6      static void Main()
7    {
8        Application.Run(new Form1());
9    }
10 }
```

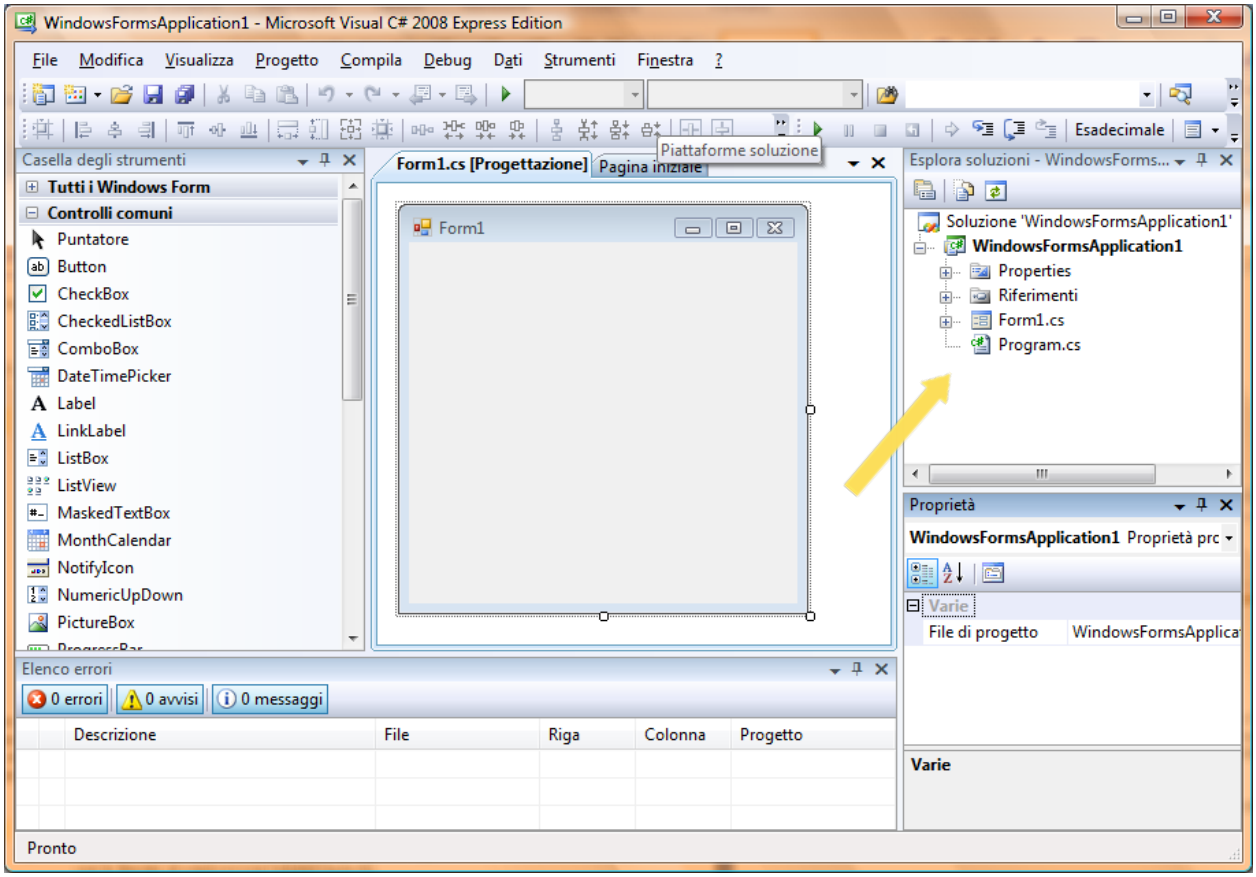
Questa è la prima applicazione Windows! Prima di esaminare il codice osserva che il listato è veramente semplice, eppure questo codice genera una finestra che possiede tutte le principali caratteristiche che conosciamo in ambiente Windows: in particolare sono presenti i tre pulsanti in alto a destra (riduci ad icona, ingrandisci/ripristina, chiudi) ed inoltre è possibile spostare e ridimensionare la finestra stessa.

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 2	uso il namespace System.Windows.Forms per gestire applicazioni che utilizzano i Form, cioè le finestre di windows con i relativi controlli;
riga 4	la classe form1, contenitore dei metodi della Nostra applicazione, deriva dalla classe Form (vediamo infatti scritto class Form1: Form), ciò significa che eredita e quindi le appartengono tutte le proprietà ed i comportamenti delle finestre di windows. Questa codice si può anche leggere come: Form1 è un (is a) form.
riga 8	questa è la sola istruzione del Main: si chiede di "far girare", cioè di eseguire, l'applicazione con la creazione (new) della nostra finestra Form1.

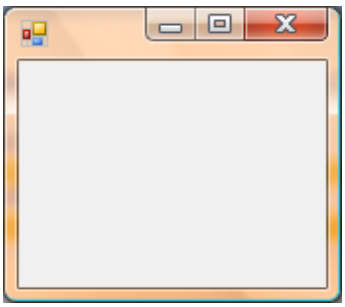
Realizza il tuo progetto

Prova a realizzare tu stesso questa applicazione: crea progetto/ Applicazione Windows Form.



Osserviamo l'ambiente e cominciamo a lavorare.
Per ora vai in Esplora soluzioni e cancella i file Form1.designer.cs e Cancellare Form1.cs (click destro sul nome del file/ Elimina), quindi preparati a scrivere il tuo programma: click destro su program.cs/visualizza codice, cancella tutto e copia il listato esaminato prima: le istruzioni indicate sono sufficienti.

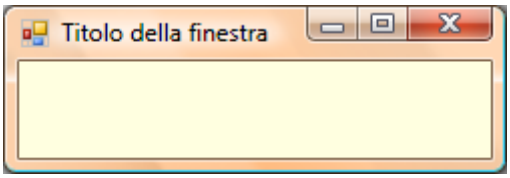
Esegui l'applicazione e verifica il regolare comportamento della finestra. Tutto OK? Complimenti!



Hai realizzato una applicazione che mostra una finestra perfettamente funzionante: è possibile ridimensionarla, spostarla, ridurla a icona e ripristinarla, infine chiuderla.

Proviamo a personalizzare un po' la nostra finestra.

Osserva e leggi il codice corrispondente



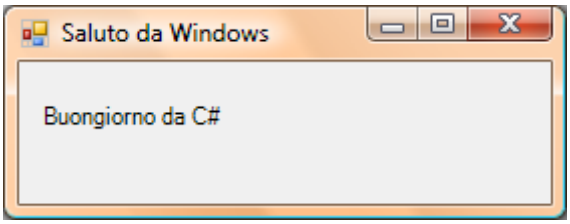

```
1  using System;
2  using System.Windows.Forms;
3  class Form1: Form
4  {
5    public Form1( )
6    {
7       this.Text = "Titolo delle finestra";
8       this.BackColor = System.Drawing.Color.LightYellow;
9    }
10
11    static void Main()
12    {
13       Application.Run(new Form1());
14    }
15 }
```

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 5	questo metodo, che ha lo stesso nome della classe, si chiama metodo costruttore, e serve per inizializzare il form che si deve costruire, serve cioè per specificare eventuali valori iniziali che vogliamo definire per determinate proprietà.
riga 7	e riga 8: nel corpo del metodo costruttore di Form1 specifichiamo in questo esempio il valore del titolo della nostra finestra ed il colore di sfondo. This si riferisce a questo oggetto, cioè in questo caso a Form1, Text e BackColor sono due proprietà a cui si accede tramite l'operatore punto (sintassi del tutto analoga a quella dei campi dei record).

6.1.1.2. Buongiorno Windows da C#: oggetto label

Osserva e leggi il codice corrispondente



La finestra della nostra applicazione è come una bacheca, sulla quale si possono posizionare oggetti di tipo diverso, ma non si può scrivere direttamente su di essa. Così per lasciare un messaggio possiamo scriverlo su un biglietto che poi fissiamo alla bacheca: in modo analogo si opera con le applicazioni Windows utilizzando un **oggetto di tipo label**.

Una label ha alcune proprietà, cioè caratteristiche particolari, a ciascuna delle quali corrisponde un valore; nello stesso modo uno studente ha alcune caratteristiche, ad esempio nome, classe, voto: sai suggerire quali potrebbero essere i valori da assegnare?

Torniamo alla label: quali sono le proprietà di una label a cui si possono assegnare i valori? Si tratta di un oggetto che ci viene messo a disposizione dal .NET Framework, quindi dovremmo andare a vedere sul manuale le specifiche tecniche. Qui una presentazione iniziale.

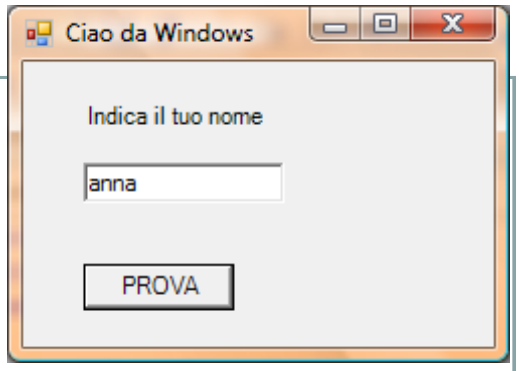
```
1  //nome file: Progam.cs
2  using System;
3  using System.Windows.Forms;
4  class Form1: Form
5  {   Label LblX;
6    public Form1( )
7    { this.Text = "Un saluto da Windows";
8       LblX = new Label();
9       LblX.Text = "Buongiorno da C#";
10      LblX.Top = 20;
11      LblX.Left = 10;
12      LblX.AutoSize = true;
13      Controls.Add(LblX);
14   }
15
16   static void Main()
17   {
18      Application.Run(new Form1());
19   }
20 }
```

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 5	dichiarazione di un oggetto di tipo label (etichetta, serve per gestire l'output: analoga al bigliettino da fissare alla bacheca, questa tecnica corrisponde al Console.WriteLine delle applicazioni Console)
riga 6	nel metodo costruttore, ove si inizializza il Form, dobbiamo gestire la nostra label;
riga 8	la label è un oggetto (un tipo di riferimento, come ad esempio i vettori) quindi è necessario allocare la memoria con l'operatore new prima di memorizzare i dati;
riga 9	proprietà Text: serve specificare il Testo che verrà visualizzato;
riga 10	e riga 11: posizione della label nel Form (il punto di coordinate 0,0 si trova in alto a sinistra);
riga 12	proprietà AutoSize: la label si dimensione automaticamente in base al testo da visualizzare;
riga 13	aggiungiamo la label all'insieme dei controlli (cioè degli oggetti) che vogliamo avere sul Form (equivale a posizionare il bigliettino sulla bacheca, dopo aver scritto il testo del messaggio).

6.1.1.3. Buongiorno personalizzato: oggetto TextBox

Osserva e leggi il codice corrispondente




```
1  //nome file: Progam.cs
2  using System;
3  using System.Windows.Forms;
4  class Form1: Form
5  {  Label LblX;
6     TextBox txtA;
7     Button btnProva;
8     public Form1( )
9     {  this.Text = "Ciao da Windows";
10
11       LblX = new Label();
12       LblX.Text = "Indica il tuo nome";
13       LblX.Top = 20;   LblX.Left = 30;   LblX.AutoSize = true; Controls.Add(LblX);
14
15       txtA = new TextBox();
16       txtA.Top = 50;  txtA.Left = 30;   Controls.Add(txtA);
17
18       btnProva = new Button();
19       btnProva.Text = "PROVA"; btnProva.Top = 100;  btnProva.Left = 30; Controls.Add(btnProva);
20    }
21
22    static void Main()
23    {
24       Application.Run(new Form1());
25    }
26 }
```

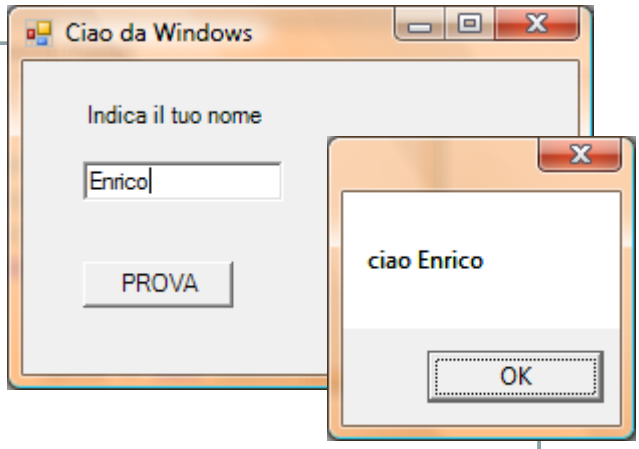
In questo esempio sono presenti due nuovi controlli: un TextBox per l'inserimento dei dati da tastiera ed un pulsante: se si prova però ad effettuare un click sul bottone, non succede nulla!

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 6	dichiarazione di un oggetto di tipo TextBox (casella per l'inserimento di dati da tastiera, serve per gestire l'input: corrisponde al Console.ReadLine delle applicazioni Console)
riga 7	dichiarazione di un oggetto di tipo Button, serve per l'interazione on l'utente attraverso la gestione degli eventi (es. click del mouse);
riga 15	e riga 16: creo la nuova TextBox, assegno i valori alle proprietà e la aggiungo ai controlli;
riga 18	e riga 19: creo il nuovo pulsante, assegno i valori alle proprietà e lo aggiungo ai controlli.

6.1.1.4. Gestione eventi e MessageBox

Osserva e leggi il codice corrispondente




```
1  //nome file: Progam.cs
2  using System;
3  using System.Windows.Forms;
4  class Form1: Form
5  {
6     Label LblX;
7     TextBox txtA;
8     Button btnProva;
9     public Form1( )
10    {  this.Text = "Ciao da Windows";
11       LblX = new Label();
12       LblX.Text = "Indica il tuo nome";
13       LblX.Top = 20;   LblX.Left = 30;   LblX.AutoSize = true; Controls.Add(LblX);
14
15       txtA = new TextBox();
16       txtA.Top = 50;  txtA.Left = 30;   Controls.Add(txtA);
17
18       btnProva = new Button();
19       btnProva.Text = "PROVA"; btnProva.Top = 100;  btnProva.Left = 30;
20       btnProva.Click += new EventHandler(btnProva_Click);
21       Controls.Add(btnProva);
22    }
23
24    static void Main()
25    {
26       Application.Run(new Form1());
27    }
28
29    void btnProva_Click(object sender, EventArgs e)
30    {
31       MessageBox.Show("Ciao "+txtA.Text);
32    }
33 }
```

Analizziamo in dettaglio il listato, indicando man mano le istruzioni che vengono eseguite

riga 20	viene gestito l'evento Click sul pulsante, in particolare ci si collega al metodo btnProva_Click, che dovremo definire di seguito, e che specifica la operazioni che la nostra applicazione deve compiere in risposta alla pressione del tasto destro del mouse da parte dell'utente;
riga 29	intestazione del metodo btnProva_Click che gestisce il click sinistro del mouse (come parametri riceve l'oggetto da cui deriva l'evento -pulsante btnProva- ed i dati sull'evento –click sinistro in una posizione che corrisponde a quella del pulsante-) ;
riga 31	istruzione da eseguire per gestire l'evento click: mostra un messaggio di saluto personalizzato.

6.1.1.5. Visualizzazione di una immagine

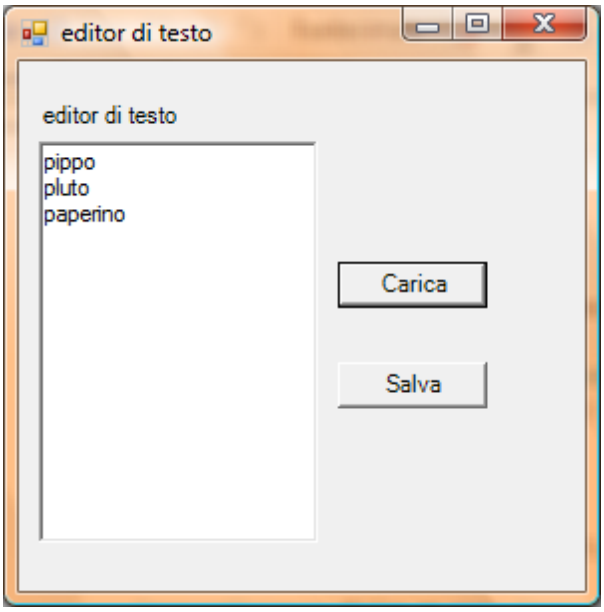
Osserva e leggi il codice corrispondente


```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsApplication1
{
   public class Form1 : System.Windows.Forms.Form
   {
      Label LblX;
      PictureBox picImmagine;

      public Form1()
      {
         Text = "immagine";

         LblX = new Label();
         LblX.Text = "visualizza immagine";
         LblX.Top = 20;
         LblX.Left = 10;
         LblX.AutoSize = true;
         Controls.Add(LblX);

         PictureBox picImmagine = new PictureBox();
         picImmagine.Location = new Point(10, 40);
         picImmagine.Size = new Size(260, 200); //LARGHEZZA, ALTEZZA
         picImmagine.SizeMode = PictureBoxSizeMode.StretchImage;
         picImmagine.Image = Image.FromFile(@"D:\C#\test.jpg");

         Controls.Add(picImmagine);
      }


      static void Main()
      {
         Application.Run(new Form1());
      }

   }    // end class Form1

}
```

Qui abbiamo un oggetto nuovo: la PictureBox serve a visualizzare un'immagine su un Form. Osserva come rimane sempre uguale la parte relativa alla gestione della finestra! Ciò significa che moto si può realizzare semplicemente utilizzando gli oggetti che ci vengono messi a disposizione dal .NET Framework.

6.1.1.6. Editor di testo

Osserva e leggi il codice corrispondente




```
using System;
using System.IO;
using System.Windows.Forms;
public class Form1 : Form
  {  Label LblX;
     TextBox txtA;
     Button btnCarica;
     Button btnSalva;
     public Form1()
     {  Text = "editor di testo";
        LblX = new Label();  // label
        LblX.Text = "editor di testo";
        LblX.Top = 20;
        LblX.Left = 10;
        LblX.AutoSize = true;
        Controls.Add(LblX);
        btnCarica = new Button(); // bottone per leggere i dati dal file e visualizzarli nella TextBox
        btnCarica.Text = "Carica";    btnCarica.Top = 100;  btnCarica.Left = 160;
        btnCarica.Click += new EventHandler(btnCarica_Click);
        Controls.Add(btnCarica);
        btnSalva = new Button();// bottone per memorizzare sul file il contenuto della  TextBox
        btnSalva.Text = "Salva";   btnSalva.Top = 150;btnSalva.Left = 160;
        btnSalva.Click += new EventHandler(btnSalva_Click);
        Controls.Add(btnSalva);
        txtA = new TextBox();  // TextBox per scrivere liberamente il nostro testo
        txtA.Top = 40; txtA.Left = 10;   txtA.Size = new System.Drawing.Size(140, 200);
        txtA.Multiline = true;
        Controls.Add(txtA);
     }

     void btnCarica_Click(object sender, EventArgs e)
     {  StreamReader file = new StreamReader(@"d:\c#\test.txt");
        txtA.Text = file.ReadToEnd();
        file.Close();
     }

     void btnSalva_Click(object sender, EventArgs e)
     {  StreamWriter file = new StreamWriter(@"d:\c#\test.txt");
        file.Write(txtA.Text);
        file.Close();
     }
     static void Main()
     {
        Application.Run(new Form1());
     }
  }    // end class Form1
```

6.2. PROGRAMMAZIONE VISUALE

PREREQUISITI

Conoscere gli elementi di base della programmazione in linguaggio C#

OBIETTIVI

Saper realizzare semplici applicazioni windows avvalendosi delle tecniche di programmazione visuale

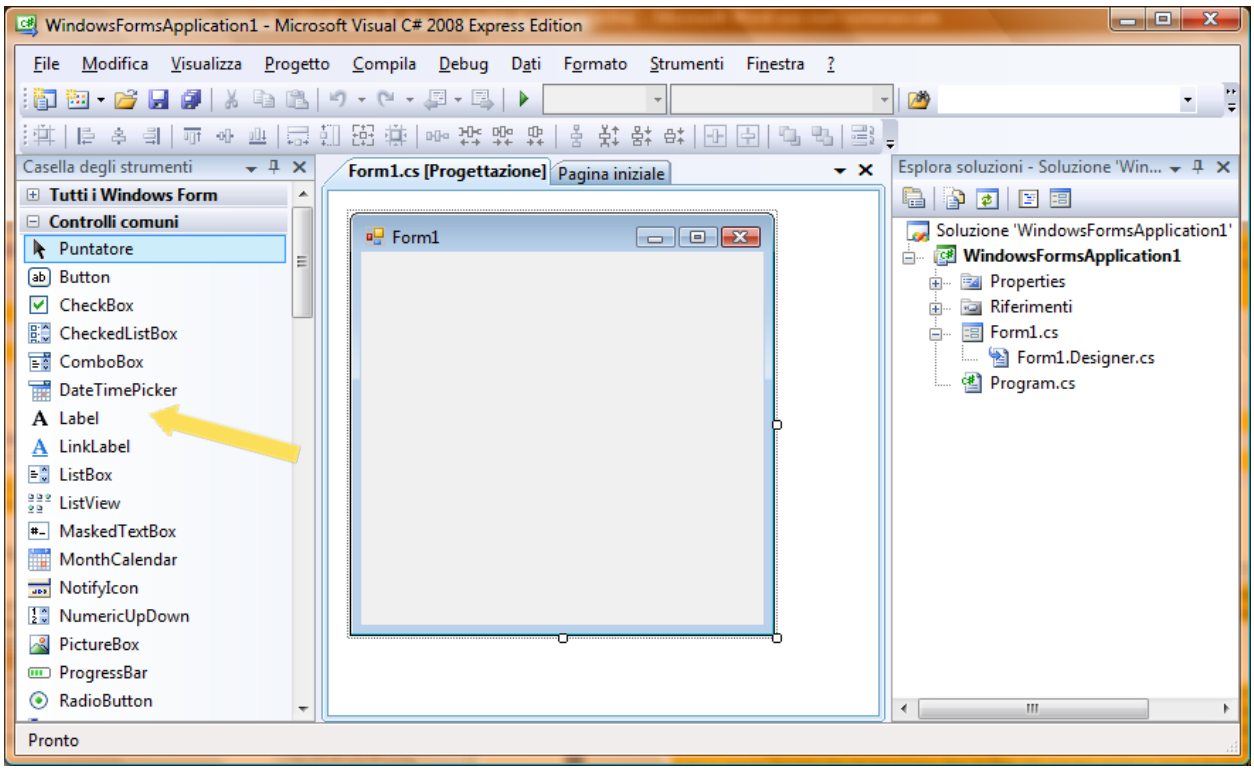
6.2.1. Introduzione alla programmazione visuale

Introduzione alla programmazione visuale.

6.2.1.1. Visual C# Express Edition 2008 - Ambiente di lavoro visuale

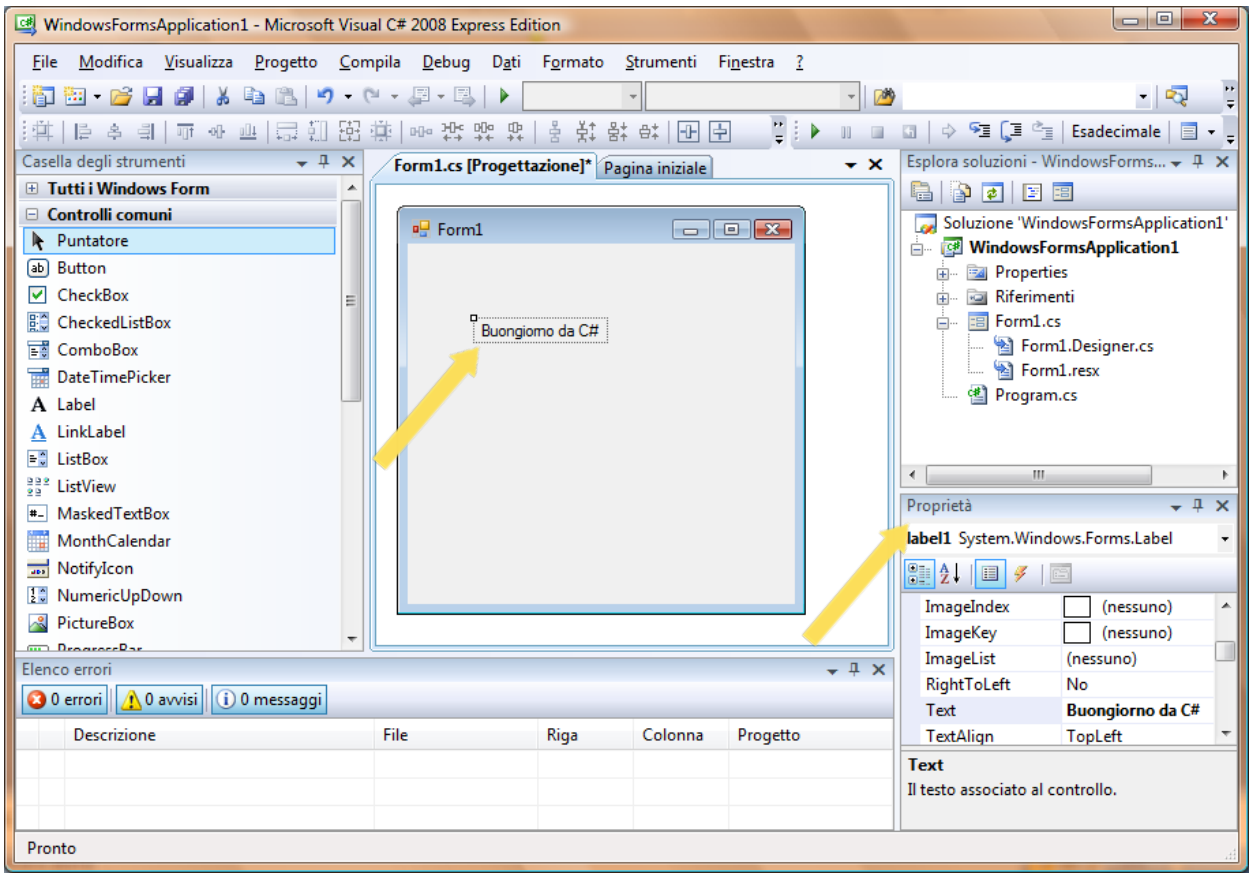
Realizziamo ora una applicazione utilizzando l'**ambiente di sviluppo visuale**: al solito Crea/Progetto/ Applicazione Windows Form e otteniamo la videata in figura (eventualmente vai su Menu Visualizza/Esplora Soluzioni , Visualizza/Finestra Proprietà e ancora Visualizza/Casella degli strumenti.

In Esplora soluzioni osserviamo la struttura del progetto e vediamo che sono presenti tre file con estensione cs, che andremo ad esaminare. Al centro la scheda Form1.cs in **modalità Progettazione** (per vedere il codice corrispondente F7) , a sinistra la casella degli strumenti.

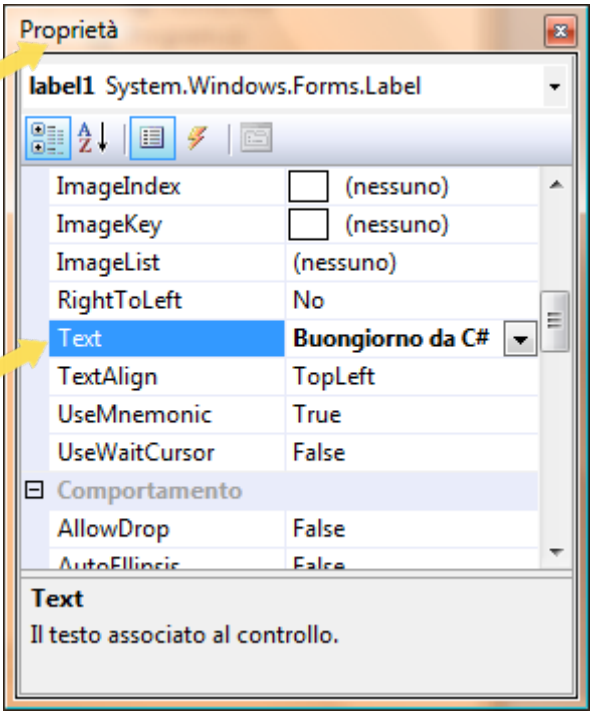


Prova ad inserire una **label** utilizzando la **casella degli strumenti** (click sinistro sull'oggetto label e trascinamento sul form, oppure doppio click): in basso a destra si apre la **finestra delle proprietà** della label stessa (in genere vengono mostrare le **proprietà dell'oggetto selezionato**).

Riconosci le proprietà che abbiamo gestito a livello di codice nel capitolo precedente, ora invece semplicemente inseriamo i valori che ci interessano : prova a modificare la **proprietà Text**.



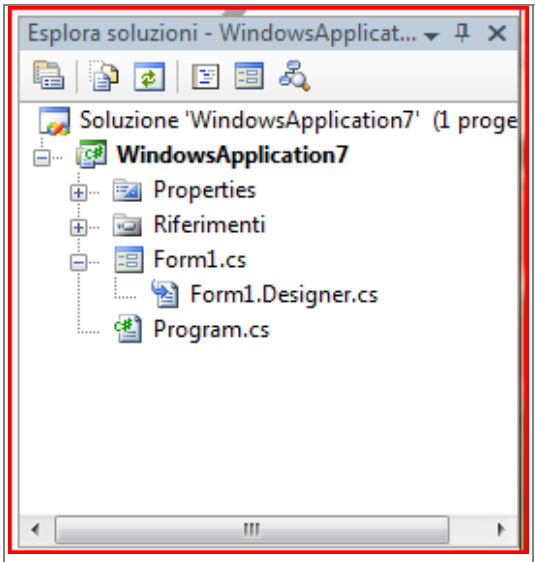
Avvia il debug (F5) e tutto automaticamente funziona. **Hai realizzato la tua prima applicazione con metodo visuale, bene!**



Il sistema ha provveduto a scrivere il codice al nostro posto, se vuoi puoi andare a controllare! È un po' più complesso di quanto mostrato prima, ma solo perché è necessario anche gestire tutto lo sviluppo visuale.

6.2.1.2. Applicazione realizzata con metodo visuale: cosa c'è dietro?

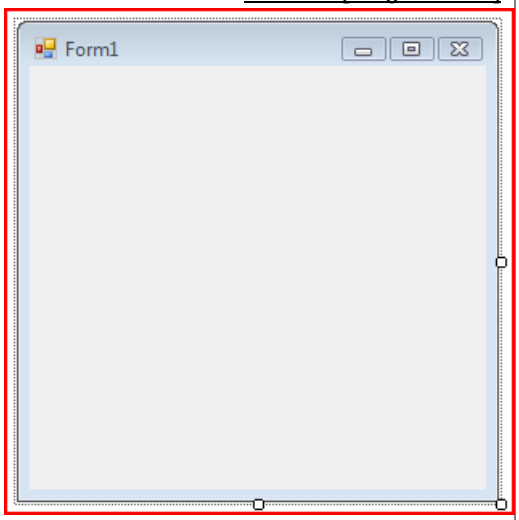
All'avvio il nuovo progetto si presenta come segue:



Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsApplication7
{
   static class Program
   {
      /// <summary>
      /// Punto di ingresso principale dell'applicazione.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}
```

Form1.cs[Progettazione]

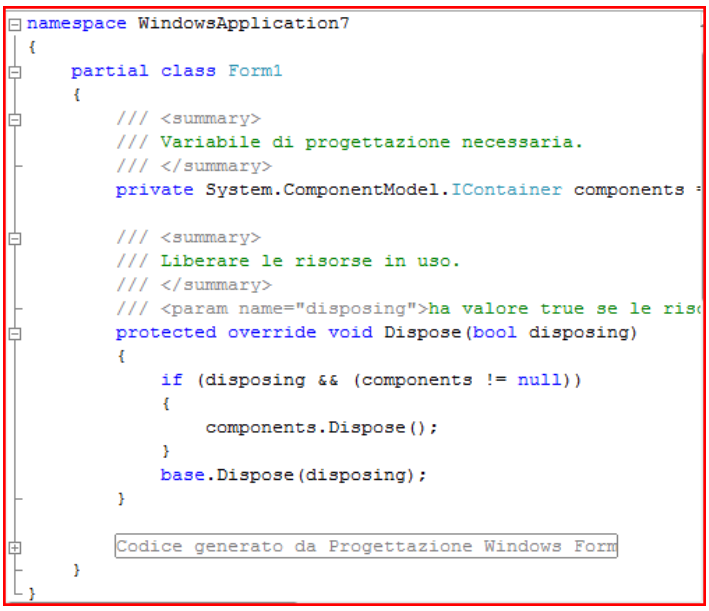


Form1.cs

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace WindowsApplication7
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
   }
}
```

Form1.Designer.cs

```
namespace WindowsApplication7
{   partial class Form1
   {
      /// <summary>
      /// Variabile di progettazione necessaria.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Liberare le risorse in uso.
      /// </summary>
      /// <param name="disposing">ha valore true se le risorse gestite devono essere eliminate, false in caso contrario.</param>
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

      #region Codice generato da Progettazione Windows Form

      /// <summary>
      /// Metodo necessario per il supporto della finestra di progettazione. Non modificare
      /// il contenuto del metodo con l'editor di codice.
      /// </summary>
      private void InitializeComponent()
      {
         this.components = new System.ComponentModel.Container();
         this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
         this.Text = "Form1";
      }

      #endregion
   }
}
```



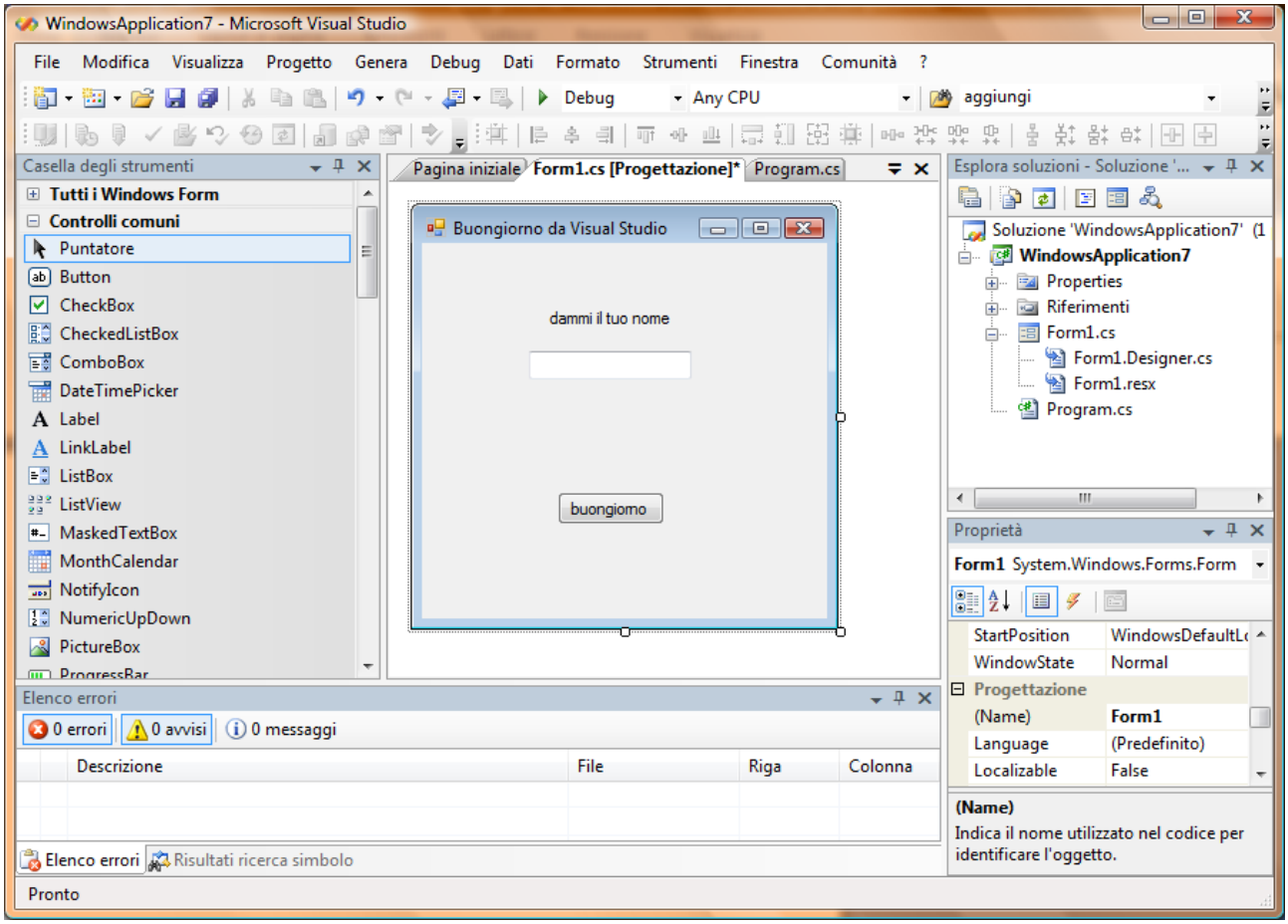
Proviamo ad aggiungere una label in visuale, e osserviamo le modifiche al codice.

Form1.Designe

```
namespace WindowsApplication7
{   partial class Form1
   {  /// <summary>
      /// Variabile di progettazione necessaria.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Liberare le risorse in uso.
      /// </summary>
      /// <param name="disposing">ha valore true se le risorse gestite devono essere eliminate, false in
     /// caso contrario.</param>
      protected override void Dispose(bool disposing)
      {  if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }
      #region Codice generato da Progettazione Windows Form

      /// <summary>
      /// Metodo necessario per il supporto della finestra di progettazione. Non modificare
      /// il contenuto del metodo con l'editor di codice.
      /// </summary>
      private void InitializeComponent()
         this.label1 = new System.Windows.Forms.Label();
         this.SuspendLayout();
         // label1
         this.label1.AutoSize = true;
         this.label1.Location = new System.Drawing.Point(58, 62);
         this.label1.Name = "label1";
         this.label1.Size = new System.Drawing.Size(167, 13);
         this.label1.TabIndex = 0;
         this.label1.Text = "Buongiorno da C#";
         //
         // Form1
         this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
         this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
         this.ClientSize = new System.Drawing.Size(284, 264);
         this.Controls.Add(this.label1);
         this.Name = "Form1";
         this.Text = "Form1";
         this.ResumeLayout(false);
         this.PerformLayout();
      }
      #endregion
      private System.Windows.Forms.Label ;
   }
}
```

6.2.1.3. Aggiungiamo altri controlli: Label, TextBox, Button

Form1.cs[Progettazione]



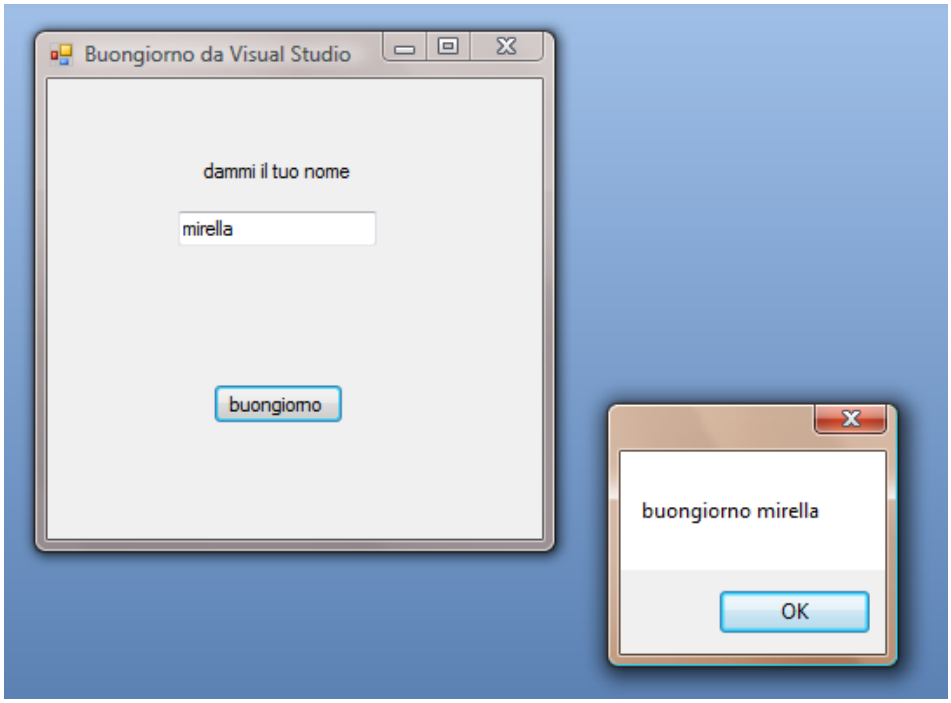
Form1.cs Codice

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsApplication7
{  public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         MessageBox.Show("buongiorno " + textBox1.Text);
      }
   }
}
```

Form1.Designer.cs

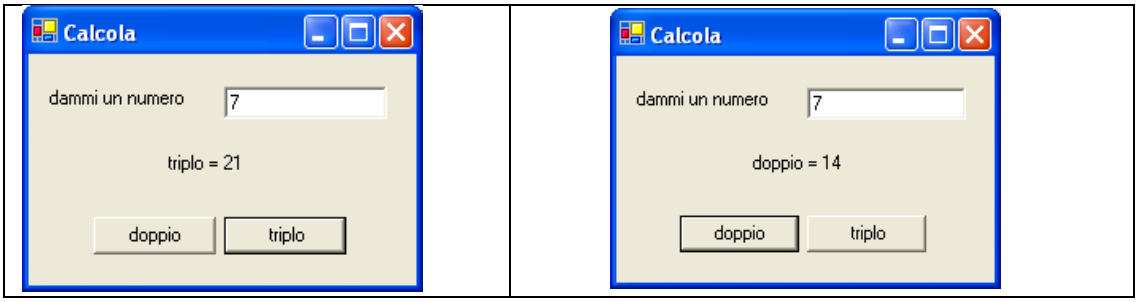
```
namespace WindowsApplication7
{  partial class Form1
   {
      /// <summary>
      /// Variabile di progettazione necessaria.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Liberare le risorse in uso.
      /// </summary>
      /// <param name="disposing">ha valore true se le risorse gestite devono essere eliminate, false in
caso contrario.</param>
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }
      #region Codice generato da Progettazione Windows Form
      /// <summary>
      /// Metodo necessario per il supporto della finestra di progettazione. Non modificare
      /// il contenuto del metodo con l'editor di codice.
      /// </summary>
      private void InitializeComponent()
      {
         this.label1 = new System.Windows.Forms.Label();
         this.button1 = new System.Windows.Forms.Button();
         this.textBox1 = new System.Windows.Forms.TextBox();
         this.SuspendLayout();
         //
         // label1
         //
         this.label1.AutoSize = true;
         this.label1.Location = new System.Drawing.Point(87, 46);
         this.label1.Name = "label1";
         this.label1.Size = new System.Drawing.Size(91, 13);
         this.label1.TabIndex = 0;
         this.label1.Text = "dammi il tuo nome";
         //
         // button1
         this.button1.Location = new System.Drawing.Point(95, 175);
         this.button1.Name = "button1";
         this.button1.Size = new System.Drawing.Size(75, 23);
         this.button1.TabIndex = 1;
         this.button1.Text = "buongiorno";
         this.button1.UseVisualStyleBackColor = true;
```

```
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(75, 76);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(114, 20);
            this.textBox1.TabIndex = 2;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(284, 264);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.label1);
            this.Name = "Form1";
            this.Text = "Buongiorno da Visual Studio";
            this.ResumeLayout(false);
            this.PerformLayout();
        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TextBox textBox1;
    }
}
```



6.2.1.4. Calcolatrice

Problema→ realizzare una applicazione che svolga le funzioni rappresentate in figura



Riconosci i controlli da utilizzare:

- label: una per la richiesat del numero, un'altra per la visualizzazionde del risutat
- TextBox: per l' inserimento del numero
- Button: due bottoni, associati alle due diverse elaborazioni

Buon lavoro!

HAI IMPARATO A ...

1. Sviluppare una semplice applicazione windows
2. Utilizzare un ambiete di sviluppo visuale

SITOGRAFIA

Nome del sito - http://www.italy.fsfeurope.org/index.it.html
Portale della La Free Software Foundation Europe (FSFE), organizzazione senza fini di lucro dedicata al Software Libero.

Nome del sito - http://www.microsoft.com/italy/beit/
Microsoft per sviluppatori, professionisti IT e web designer: è suddiviso in due sezioni dedicate agli sviluppatori ed ai professionisti IT e una grande quantità di **web cast tecnici**

Nome del sito http://msdn.microsoft.com/it-it/library/default.aspx
risorsa indispensabile per gli sviluppatori che usano strumenti, prodotti e tecnologie Microsoft. Contiene una vasta **documentazione tecnica sulla programmazione**, che include codice di esempio, documenti, articoli tecnici e guide di riferimento

Nome del sito - http://www.icsharpcode.net/OpenSource/SD/
Sito di riferimento per **SharpDevelop** che è un ambiente di sviluppo free per C# e altri linguaggi sulla piattaforma .NET di Microsoft. É un software open-source ed è disponibile da questo sito il download sia del codice sorgente, sia dell' eseguibile.

Nome del sito http://xhtml.html.it/guide/leggi/51/guida-html/
Manuale HTML, ma non solo! Si possono trovare utili manuali di riferimento su tutti i principali campi di interesse per chi si occupa di informatica.