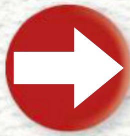
Modulo 2

Introduzione alla programmazione

Unità didattiche

1. Il problem solving
2. Progettare algoritmi per il problem solving



CHE COSA IMPAREREMO...

1. Gli obiettivi e gli strumenti per la programmazione
2. La classificazione dei linguaggi di programmazione
3. Gli obiettivi e gli strumenti per il problem solving
4. La progettazione di algoritmi mediante flow chart
5. La progettazione di algoritmi mediante pseudocodice

AULADIGITALE

MATERIALI DIGITALI PER LO STUDENTE

- Test in autovalutazione
- Approfondimenti

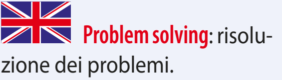
© RCS LIBRI EDUCATION SPA

Il problem solving

1.1 Gli obiettivi della programmazione

L'informatica s'interessa del trattamento automatico dell'informazione; il computer è lo strumento con cui l'uomo interagisce per automatizzare i processi e risolvere i problemi reali. La gestione dei processi aziendali, decisionali e operativi viene oggi ampiamente supportata da macchine che svolgono in modo automatico ed efficiente numerose funzioni sostituendosi all'uomo.
Il funzionamento dei computer richiede la predisposizione di programmi, cioè di istruzioni che ne guidano l'operatività.

I **programmatori** sono esperti informatici che affrontano problematiche reali elaborando delle soluzioni mediante l'uso del computer.



Problem solving: risoluzione dei problemi.

L'attività del programmatore è legata ai settori della scienza e della ricerca che analizzano strategie e metodologie per affrontare la risoluzione dei problemi sfruttando l'intelligenza umana; questa attività prende il nome di **problem solving**.

© RCS LIBRI EDUCATION SPA

Il **problem solving** è una tecnica che, attraverso l'insieme dei processi messi in atto, permette di analizzare, affrontare e risolvere problemi del mondo reale.

La programmazione persegue l'obiettivo di sfruttare in modo intelligente le grandi potenzialità del computer attraverso l'**interazione uomo-macchina**, processo nel quale l'utente comune o l'esperto comunicano con il computer.

I computer moderni offrono la possibilità di:
- memorizzare enormi quantità di dati;
- rielaborare dati in modo rapido ed efficiente;
- interagire in tempo reale con gli utenti finali.

La programmazione in ambito aziendale ha come obiettivo quello di sfruttare al meglio le risorse a disposizione per la risoluzione dei problemi gestionali. Questa problematica coinvolge sia la scelta dei metodi di memorizzazione e accesso ai dati al fine di un'interazione efficace con la macchina sia la scelta della tipologia e dell'ordine dei comandi da impartire al computer.

L'attività di risoluzione dei problemi applicata al computer sfocia nella stesura di **programmi**.

Un **programma** è una sequenza di istruzioni avente l'obiettivo di fornire una procedura automatica di risoluzione di un problema mediante l'uso del computer.

Il programma viene successivamente utilizzato dagli **utenti finali**, che interagiscono con la macchina per ottenere le informazioni desiderate; essi usano il programma come strumento per la risoluzione di problemi affini a quello per cui il programma è stato creato.

La sequenza di passi logici adottati per il raggiungimento del risultato desiderato è denominata **algoritmo**. Gli algoritmi mirano a trovare una **soluzione ottimale** al problema in oggetto, seguendo la procedura migliore per arrivare alla soluzione nel modo più **efficiente** possibile.

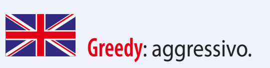
L'**efficienza** misura l'impiego ottimale delle risorse nella risoluzione del problema.

L'efficienza può riguardare differenti aspetti, tra cui la *rapidità*: un algoritmo efficiente deve essere rapido perché si vuole pervenire al più presto alla soluzione.

In altri casi l'efficienza si misura principalmente in termini di *precisione* del risultato: per esempio, trovare una soluzione numerica approssimata a un problema con il numero di cifre decimali più alto possibile. In tal caso può essere necessario rinunciare in parte all'esigenza di rapidità.
In generale, la soluzione efficiente deriva da una ricerca dell'equilibrio ottimale tra precisione e rapidità.

Nonostante i grandi progressi compiuti dalla scienza informatica non esistono tuttora regole e indicazioni assolute che permettano la scrittura di *algoritmi corretti* per la soluzione di qualunque tipo di problemi in quanto:
- la gamma di problemi da affrontare è tale da rendere impossibile una qualunque classificazione esaustiva;
- spesso risulta impossibile definire in modo rigoroso l'efficacia e l'efficienza dei programmi, in quanto, per molti aspetti, la valutazione è legata a criteri soggettivi (per esempio, la generalità, la semplicità, la facilità d'uso ecc.);
- la potenza di calcolo, anche mediante l'uso dei più recenti e avanzati computer, mostra comunque dei limiti.

La complessità di alcuni problemi è tale da rendere impossibile, in tempi e con modalità ritenute accettabili, la ricerca della soluzione ottimale. Si adottano dunque dei procedimenti, definiti **algoritmi euristici**, che raggiungono soluzioni approssimate, cioè *le più vicine possibili* alla soluzione ottimale.


Greedy: aggressivo.

Classici esempi di algoritmi euristici applicati nella risoluzione di problemi complessi sono gli **algoritmi *greedy***.

Un **algoritmo *greedy*** è un algoritmo che ricerca un risultato ottimale attraverso la scelta della soluzione più appetibile ottenuta analizzando un sottoinsieme di possibili soluzioni. Questa tecnica consente, in alcuni casi, di ottenere soluzioni quasi ottimali a problemi complessi.

Problema dello zaino: problema di ottimizzazione combinatoria, noto anche con la sigla KP, che trova diverse applicazioni reali.

ESEMPIO **Il problema dello zaino (*Knapsack Problem*)**

Per realizzare una traversata un alpinista deve preparare uno zaino, che non può superare un determinato peso massimo (X kg). Degli oggetti O_1, O_2, …, O_n disponibili l'alpinista deve **scegliere il sottoinsieme migliore da caricare nel suo zaino**, conoscendo:

- il peso P_i di ciascun oggetto O_i;
- l'importanza W_i di ciascun oggetto O_i.

Soluzione

Il problema richiede di identificare un sottoinsieme di oggetti tra quelli a disposizione tale che la somma dei loro pesi sia uguale o inferiore a X kg e la somma della loro importanza sia massima. Nonostante la formulazione del problema sia semplice e intuitiva, la soluzione può non essere immediata; infatti è necessario considerare tutte le possibili combinazioni distinte di oggetti che soddisfano la condizione sul peso totale e valutarne l'importanza complessiva, al fine di identificare la combinazione migliore.

Se il numero di oggetti è elevato, il problema diventa di complessa soluzione; con $n = 10$ oggetti bisogna considerare $2^{10} = 1024$ possibili combinazioni diverse e valutare peso e importanza di ciascuna di esse.

Si possono adottare quindi approcci *greedy* per la soluzione di questo problema; per esempio:
– si sceglie l'oggetto di maggiore importanza il cui peso totale non superi il limite massimo di X kg e lo si include nello zaino;
– si ripete il procedimento sulla restante parte degli oggetti finché non siano più inseribili nuovi oggetti per limiti di peso.

1.2 I linguaggi di programmazione

La comunicazione verbale e scritta tra persone si basa su **linguaggi naturali** che sono caratterizzati da proprietà comunemente note e definite:

- proprietà *lessicali*, relative all'uso di parole e locuzioni (frasi o proposizioni);
- proprietà *sintattiche*, che riguardano le regole e le modalità con cui si combinano le parole per formare le frasi;
- proprietà *semantiche*, che riguardano la relazione che intercorre tra le singole parole e frasi e i concetti-significati a cui si riferiscono.

ESEMPIO Si consideri le seguenti parole in lingua inglese (*lessico*):

the – is – table – pencil – on – the

Insieme costituiscono una frase se disposte nell'ordine giusto secondo la *sintassi* della lingua inglese:

the pencil is on the table

Il significato della frase è comprensibile senza incertezze a chiunque conosca l'inglese (*semantica*).

In italiano le due frasi "*Io non penso proprio*" e "*Proprio io non penso*" utilizzano lo stesso lessico, ma trasmettono un significato diverso grazie ad una diversa disposizione delle parole.

Spesso, specialmente in ambito tecnico-scientifico, per raggiungere una comunicazione efficace si rende necessario l'uso di formalismi, cioè linguaggi specifici molto precisi e delimitati, il più possibile privi di ambiguità. Si tratta di linguaggi definiti, per contrasto con i linguaggi naturali, *linguaggi artificiali*.
Ci si avvicina così alle caratteristiche di un linguaggio che consenta un'elaborazione automatica.

ESEMPIO Il diagramma Entità-Relazione usato per rappresentare graficamente lo schema concettuale di una base di dati è un esempio di formalismo.

Sono di questo tipo i linguaggi usati nella comunicazione uomo-macchina per la risoluzione di problemi mediante l'uso del computer, si tratta dei *linguaggi di programmazione*.

Un **linguaggio di programmazione** è un linguaggio formale, dotato di lessico, sintassi e semantica ben definiti, utilizzabile per il controllo e la gestione di un computer.

La programmazione basata su uno specifico linguaggio si realizza attraverso la scrittura di un programma sotto forma di codice. Il codice racchiude sequenze di **istruzioni** (o **comandi**).

Il *lessico* del linguaggio di programmazione comprende un insieme di parole chiave che identificano alcune operazioni basilari conosciute dalla macchina. La *sintassi* specifica la modalità e la forma con cui tali comandi vanno impartiti alla macchina. La *semantica* abbina a ciascun comando un significato, che si traduce in un concetto chiaro e condiviso per gli informatici che lo utilizzano.

1.3 Il problem solving applicato alla programmazione

I computer sono in grado di eseguire in modo efficiente sequenze di istruzioni, denominate programmi, codificate da un programmatore mediante uno specifico linguaggio di programmazione.

Un programma è quindi la traduzione, mediante un linguaggio di programmazione scelto dall'esperto, dei passi di un algoritmo ideato per la soluzione di un determinato problema. L'algoritmo progettato dall'esperto informatico viene cioè *codificato* o *implementato* in uno specifico linguaggio di programmazione.

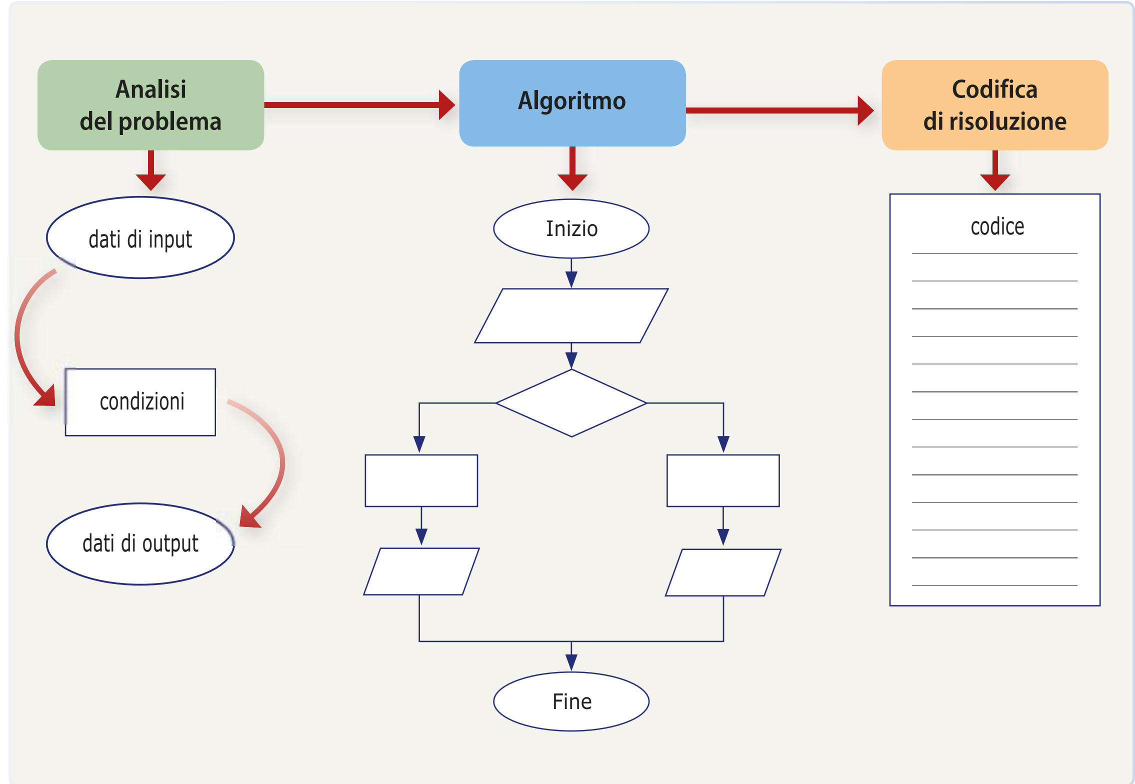
Codificare o **implementare** significa tradurre un algoritmo in una specifica codifica, in base a un linguaggio di programmazione prescelto.

I passi concettuali di approccio al problem solving mediante il supporto informatico sono:
1. l'*analisi* del problema;
2. la *progettazione* di un algoritmo che fornisce una risoluzione logica del problema;
3. la *codifica* (implementazione) dell'algoritmo in uno specifico linguaggio di programmazione, ovvero un linguaggio comprensibile ed eseguibile da un elaboratore.

L'*analisi* del problema individua i punti di partenza e di arrivo per la sua risoluzione e le condizioni che devono essere soddisfatte.

La *progettazione* algoritmica propone una soluzione al problema che soddisfi le condizioni individuate al punto precedente.

La *codifica* (*implementazione*) traduce l'algoritmo nel programma mediante uno specifico linguaggio di programmazione.



I concetti fondamentali della programmazione e dell'approccio algoritmico sono indipendenti dal linguaggio adottato.
L'individuazione di un algoritmo risolutivo di un determinato problema reale costituisce un momento concettualmente distinto dalla scelta del linguaggio di programmazione e dalla implementazione.
Usare linguaggi di programmazione diversi per codificare lo stesso algoritmo è come usare lingue diverse per dire la stessa cosa.

Nei prossimi paragrafi saranno trattati i concetti base relativi a ciascuno dei passi precedentemente sopra definiti; nell'Unità successiva verranno affrontati più dettagliatamente l'analisi del problema e la progettazione di un algoritmo mediante diagrammi di flusso (i flow chart) e pseudocodice.

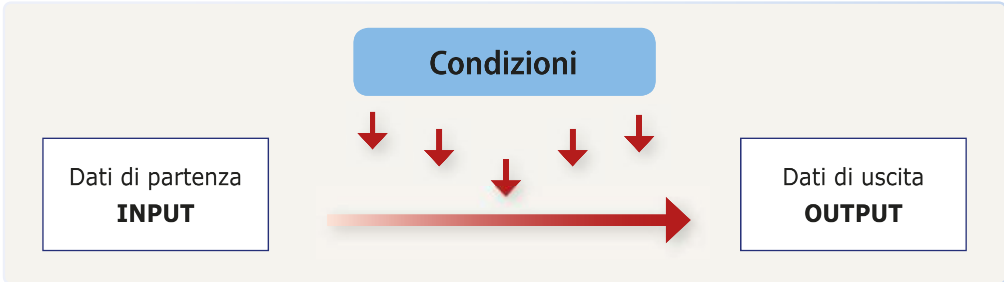
Analisi del problema

L'analisi del problema consiste nell'interpretare il problema e nell'identificare:
1. i *dati di partenza* (input);
2. i *dati di uscita* (output);
3. le *condizioni* sui dati e sui risultati.

I *dati di partenza* rappresentano la conoscenza iniziale da cui si parte per affrontare il problema.

I *dati in uscita* rappresentano l'insieme dei risultati che si desiderano ottenere.

Le *condizioni* descrivono le relazioni che devono intercorrere tra i dati in ingresso e in uscita e restringono il campo dei risultati accettabili.



ESEMPIO Riprendiamo il problema dello zaino: il peso e l'importanza di ciascun oggetto insieme al peso massimo dello zaino rappresentano i dati in ingresso; l'elenco degli oggetti costituisce l'output dell'algoritmo, mentre le condizioni imposte sul peso totale e sull'importanza massima degli oggetti selezionati rappresentano le condizioni del problema.

Chi affronta un nuovo problema con la metodologia dell'informatico si pone le seguenti domande:

Domande		Operazione
1	Quali sono i dati di partenza del problema?	Individuazione degli input.
2	Quali sono i dati di uscita che mi aspetto di ottenere?	Individuazione degli output.
3	Quali legami devono sussistere tra i dati in ingresso e quelli in uscita?	Individuazione delle condizioni del problema.

Progettazione di un algoritmo

Il passo successivo consiste nell'individuazione della strategia di soluzione, rappresentata da un algoritmo composto da sequenze di operazioni da compiere sui dati di partenza. Le operazioni eseguite sui dati d'ingresso producono i dati in uscita.

In questo passaggio non si considera l'aspetto *implementativo* (o *codifica*), ovvero la scelta del linguaggio di programmazione più idoneo per realizzare il programma; si studia invece la strategia di risoluzione migliore del problema in esame.

Lo sviluppo dell'algoritmo deve prevedere un insieme di istruzioni da eseguire con una sequenza temporale ben definita, in grado di produrre dai dati iniziali le uscite desiderate, rispettando i vincoli imposti dalle condizioni.
Le proprietà delle istruzioni incluse in un algoritmo sono le seguenti:
1. *finitezza*;
2. *non ambiguità*;
3. *generalità*.

La *finitezza* indica che ogni istruzione può essere eseguita un numero finito di volte.

La *non ambiguità* significa che le istruzioni non devono essere formulate in modo contraddittorio.

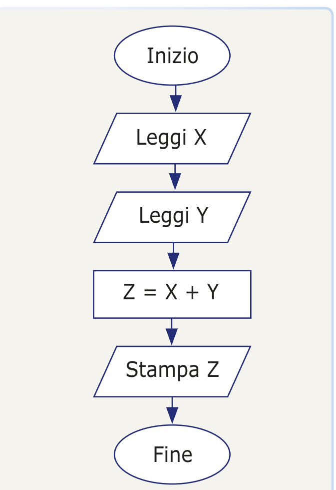
La *generalità* implica che l'algoritmo sia applicabile non solo a un insieme di dati di partenza specifico, ma a una classe di dati di ingresso generica.

ESEMPIO Un algoritmo di risoluzione del problema dello zaino deve essere applicabile a una qualunque combinazione di oggetti note le loro proprietà (peso e importanza) e un valore massimo di peso definito in modo arbitrario.

Flow chart: diagramma di flusso.

La descrizione di un algoritmo può far uso di un qualunque linguaggio formale o naturale.
Una modalità diffusa per rendere più chiara, concisa e non ambigua la descrizione è il **diagramma di flusso** che adotta un formalismo grafico per rappresentare la sequenza di istruzioni impartite.

Il **diagramma di flusso** (**flow chart**) è un formalismo grafico per rappresentare il flusso di controllo di algoritmi o procedure.

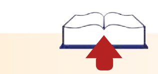


Il diagramma di flusso descrive in modo schematico i passi richiesti dall'algoritmo mediante una simbologia grafica standard; ogni tipologia di istruzioni è rappresentata con un simbolo convenzionale; i simboli, adeguatamente commentati, sono collegati mediante frecce per rappresentare il flusso dell'algoritmo, a partire dai dati d'ingresso fino ad arrivare ai dati di uscita.

L'esempio a lato mostra un primo semplice diagramma di flusso in cui due valori di ingresso (X e Y) vengono sommati per produrre in uscita il risultato dell'operazione di addizione; i diagrammi di flusso saranno approfonditi nella successiva Unità.

Una seconda modalità per rappresentare gli algoritmi è costituita dallo **pseudocodice**.

Lo **pseudocodice** (o *linguaggio di progetto*) è un linguaggio di programmazione fittizio, che non è direttamente interpretabile da un computer, ma ha lo scopo di rappresentare algoritmi, ovvero strategie di risoluzione di problemi reali.


L'esercitazione guidata 1, collocata a fine Unità, illustra come disegnare un flow chart usando il word processor Microsoft Word 2007.

Lo pseudocodice "imita" il formalismo dei linguaggi di programmazione senza vincolare l'esperto a una sintassi specifica; può essere dunque utilizzato in alternativa al diagramma di flusso per superare le limitazioni intrinseche di quest'ultimo tipo di rappresentazione.
L'esempio mostra lo pseudocodice corrispondente all'esempio di *flow chart* relativo alla somma di due valori dati in ingresso X e Y con stampa del risultato in uscita.

```
Leggi X
Leggi Y
Z = X + Y // somma i valori di X e Y
Stampa Z
```

Codifica di un algoritmo

L'ultimo passo per la risoluzione del problema nel contesto informatico è la codifica dell'algoritmo ideato per ottenere la soluzione in un linguaggio comprensibile

al computer; a tal fine si utilizzano appositi **linguaggi di programmazione** che vincolano il programmatore all'uso di un determinato lessico e di una precisa sintassi per descrivere i passi procedurali necessari.

La programmazione implica la stesura del **codice sorgente** ovvero una sequenza di istruzioni che può essere scritta mediante un qualunque programma di editing e trattamento testi (per esempio *notepad* e *wordpad* nei sistemi operativi Windows).

Esistono tuttavia programmi specifici, che forniscono un supporto grafico più avanzato per lo sviluppo di un codice sorgente, integrando funzionalità grafiche, completamento di parole, ***debugging*** ecc.

Il ***debugging*** è la procedura di verifica di correttezza del codice.

Formato binario: formato composto da una sequenza di cifre 0/1.

Piattaforma hardware: combinazione di componenti elettronici che compongono un computer.

Linguaggi di *scripting*: linguaggi che servono per il supporto e l'interazione con altri programmi più complessi.

Le istruzioni scritte in uno specifico linguaggio sono successivamente tradotte in linguaggio macchina, cioè un linguaggio comprensibile al computer; esistono due possibili strumenti utili per realizzare tale traduzione:

1. l'**interprete**, che traduce semplicemente ciascuna istruzione in linguaggio macchina ogni volta che viene eseguito il programma;
2. il **compilatore**, che genera un file in **formato binario** eseguibile direttamente sulla macchina su cui è stato compilato.

L'*interprete* necessita di altri programmi o librerie per l'esecuzione del codice e questo ne limita le prestazioni; d'altro canto una modifica al codice non vincola il programmatore a dover ricompilare tutto il codice.

La *compilazione*, demandata a un software specifico chiamato *compilatore*, genera codici eseguibili autonomamente e garantisce quindi migliori prestazioni durante l'esecuzione.Una modifica al codice comporta però una nuova compilazione. Essendo compilato per una specifica macchina, il codice eseguibile è vincolato all'hardware della macchina su cui è stato creato e potrebbe dunque non funzionare su macchine basate su hardware differenti. In linguaggio informatico si dice che il codice compilato non è, in generale, *portabile*.

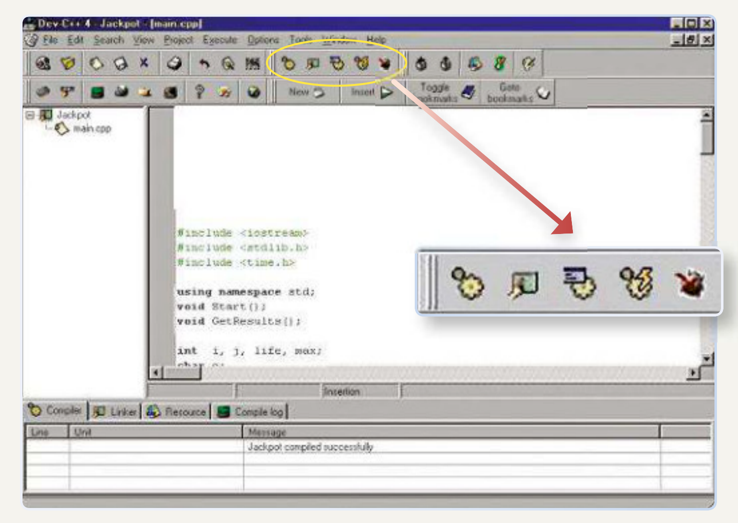
La **portabilità** è la proprietà di poter applicare il codice scritto mediante un software o uno specifico linguaggio di programmazione su diverse **piattaforme hardware.**

Per piattaforme hardware diverse si intendono computer con caratteristiche diverse oppure dispositivi come cellulari o tablet.
Per piattaforme software differenti si intendono sistemi operativi in grado di supportare il funzionamento di applicativi e software.

Per garantire la portabilità, il compilatore si avvale di un blocco denominato **linker**.

Il **linker** è un software che integra le varie parti del programma e i diversi moduli esterni, permettendo così la creazione di un'unità eseguibile autonomamente.

Esempi di linguaggi interpretati sono l'SQL e tutti i **linguaggi di *scripting*** utilizzati nel Web (per esempio, HTML e Javascript); alcuni di questi linguaggi saranno analizzati nelle prossime Unità.



Esempi di linguaggi compilati sono: C, Basic e Pascal.
Un programma avanzato di supporto alla programmazione che integra al suo interno anche compilatore, interprete, *debugger* ecc., viene solitamente denominato **IDE**.
Nell'esempio della figura a lato viene presentata la finestra principale di DevC++, un IDE per la programmazione in C++. Nella finestra centrale si scrive il codice e dalla barra dei comandi è possibile lanciare la compilazione e il debugging.

1.4 La classificazione dei linguaggi di programmazione

IDE: acronimo di *Integrated Development Environment* ovvero "Ambiente di sviluppo Integrato".

Comando primitivo: comando fornito in linguaggio macchina.

I linguaggi di programmazione, elaborati alla fine della seconda guerra mondiale, si classificano in base al livello di complessità del numero di **comandi primitivi** a cui corrisponde ciascuna istruzione.

Il linguaggio che una macchina può interpretare direttamente, senza bisogno di alcuna traduzione ulteriore, viene denominato **linguaggio macchina**; i linguaggi di programmazione vengono classificati per **livelli** in funzione della somiglianza con il linguaggio macchina.

Il **livello di un linguaggio di programmazione** viene classificato **basso** quando un linguaggio di programmazione è simile al linguaggio macchina; mentre viene classificato **alto** il livello di un linguaggio più evoluto, che richiede maggiori rielaborazioni e traduzioni per riportare le sue istruzioni al corrispettivo linguaggio macchina .

Linguaggi di basso livello

Il linguaggio di basso livello più conosciuto è *Assembler* che traduce in una nuova rappresentazione simbolica il linguaggio macchina, apportando modifiche minimali alla sequenza di istruzioni richieste dalla macchina stessa.

ESEMPIO Il programma "Hello world" in *Assembly Intel x86* (basato sul sistema operativo MS-DOS).

```
IDEAL
MODEL SMALL
STACK 100h
DATASEG
  HW   DB   "hello, world", 13, 10, '$'
CODESEG
Begin:
  MOV AX, @data
  MOV DS, AX
  MOV DX, OFFSET HW
  MOV AH, 09H
  INT 21H
  MOV AX, 4C00H
  INT 21H
```

I linguaggi di basso livello presentano alcune limitazioni:
- sono di difficile interpretazione;
- dipendono dalla macchina su cui si sta programmando;
- risultano ingestibili, in termini di tempi e complessità del codice, nella risoluzione di problemi complessi.

Il linguaggio macchina prevede infatti la specifica di ciascuna operazione primitiva necessaria per il raggiungimento dello scopo, complicando il lavoro del programmatore che ricerca soluzioni rapide ed efficaci. Inoltre, la programmazione a basso livello richiede una conoscenza approfondita delle caratteristiche hardware della macchina su cui si sta lavorando; il codice risulta specifico per la macchina su cui si programma mentre diventa, in generale, inapplicabile su macchine con hardware differenti.

La codifica *Assembler* risulta quindi *non portabile* e difficile da gestire da parte di programmatori non esperti; trova quindi applicazione in software molto specifici, in cui i programmatori hanno la necessità di interfacciarsi con un hardware non tradizionale, di cui vogliono sfruttare a pieno le potenzialità.

Linguaggi di alto livello

I linguaggi di programmazione di alto livello tendono alla costruzione di istruzioni che somigliano maggiormente al linguaggio naturale, cercando di fornire al programma un livello di astrazione superiore rispetto a quello del linguaggio macchina. Il loro obiettivo è di descrivere nel modo più semplice, conciso e intuitivo quanto il programmatore si prefigge, pur rinunciando alla possibilità di gestire in modo manuale ogni singolo evento e interazione con la macchina.

Nei linguaggi di alto livello ogni singola istruzione esprime il contenuto di un gruppo, più o meno significativo, di istruzioni in linguaggio macchina. Essi si distinguono in linguaggi interpretati e linguaggi compilati in base alle necessità di usare un interprete o un compilatore per rendere eseguibile (tradurre in linguaggio macchina) il codice.

Gli *elementi base* di un linguaggio sono:
1. le **variabili**, dati a cui si associa un'area di memoria riservata. Il modo con cui sono memorizzate le variabili dipende dal tipo di dato che si vuole immagazzinare (per esempio numero intero, numero reale, caratteri alfanumerici ecc.);
2. le **strutture dati**, strumenti di organizzazione dei dati più articolate rispetto alle singole variabili. Consentono di combinare insieme dati aventi un significato comune o un preciso ordinamento;
3. le **istruzioni**, comandi che svolgono una funzione predefinita; le istruzioni fondamentali sono quelle che permettono l'ingresso e l'uscita di dati;
4. le **strutture di controllo**, combinazioni di parole chiave che permettono di guidare il flusso del programma in base ai valori assunti da specifiche variabili o espressioni in un determinato istante. Le strutture di controllo permettono, per esempio, di specificare le espressioni condizionali del tipo "*Se... allora... altrimenti*"; i cicli come "*Ripeti... fino a...*"; i salti incondizionati "*Vai alla riga di codice numero...*".

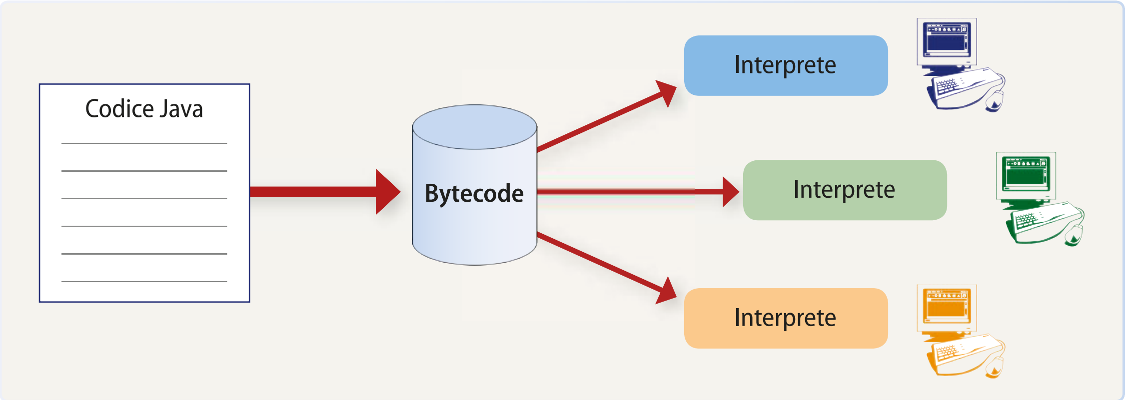
È buona norma strutturare il codice di un programma in modo da far corrispondere a ciascuna sua parte una precisa **funzione**; assegnato un nome a tale funzione, è possibile richiamarla nel programma senza doverne riscrivere il contenuto.

ESEMPIO Una funzione può essere una sequenza di istruzioni denominata "Calcolo_interesse" che calcola l'interesse da corrispondere su un finanziamento. La funzione ha un valenza generale ed è quindi applicabile quindi a qualunque problema analogo.

Programmazione a oggetti

Nell'ambito della programmazione ad alto livello, l'introduzione della **programmazione a oggetti** ha modificato radicalmente l'approccio alla programmazione. Elaborato nel 1983 a Palo Alto in California (USA), il **paradigma a oggetti** è stato applicato nella maggior parte dei linguaggi di programmazione moderni; i più famosi sono C++ e Java.

A differenza degli altri linguaggi, Java non adotta un semplice compilatore o interprete bensì un approccio ibrido che gli consente di generare un codice di livello intermedio, denominato ***bytecode***, il quale, a sua volta, viene interpretato. Questa tipologia di approccio risolve il problema della portabilità del codice, in quanto il bytecode è portabile e rende quindi Java un **linguaggio multipiattaforma**, ossia applicabile su un'ampia scala di *piattaforme hardware*. Questa proprietà lo ha reso particolarmente fruibile nella programmazione di applicazioni per cellulari e dispositivi portabili.



OOP: è l'acronimo di *Object Oriented Programming*, programmazione orientata agli oggetti.

La **programmazione orientata agli oggetti** (**OOP**) è una tecnica di programmazione che prevede la creazione di "oggetti" che interagiscono tra loro attraverso lo scambio di messaggi.
Gli oggetti aventi caratteristiche comuni vengono raggruppati in categorie, denominate *classi*; per ciascuna categoria vanno specificate delle strutture dati accessorie e delle procedure che operano su di esse.

L'**incapsulamento** è una proprietà per cui in una classe si ritrovano sia le proprie strutture dati sia le procedure che operano su di esse.

Grazie alla proprietà di incapsulamento la classe diventa un'unità a sé stante. Le caratteristiche degli oggetti vengono comunemente ridefinite come *attributi*, se relativi ai dati o *metodi*, se relativi a procedure.

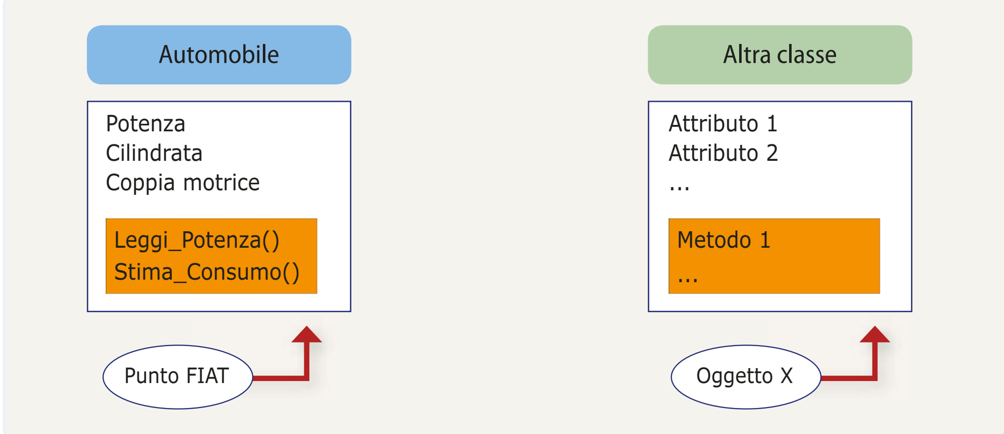
L'**ereditarietà** è una proprietà per cui le classi sono organizzate e strutturate in modo gerarchico e possono essere definite a partire da altre più generali; una classe più specifica "eredita" attributi e metodi dalla superclasse più generale a cui appartiene e la specializza con metodi e attributi propri.

ESEMPIO La classe "Animali vertebrati" è una sottoclasse di "Animali" che ne eredita le proprietà principali e possiede caratteristiche più specifiche, per esempio la presenza delle vertebre.

Le procedure scritte per una classe possono essere ridefinite, riapplicate e specializzate in una sottoclasse della classe di partenza; questa proprietà è definita **polimorfismo**.
La programmazione a oggetti permette di organizzare in modo più strutturato e semanticamente coerente le istruzioni relative ai dati e alle operazioni compiute su di esse; viene quindi limitata la *ridondanza del codice*, semplificando il lavoro del programmatore.

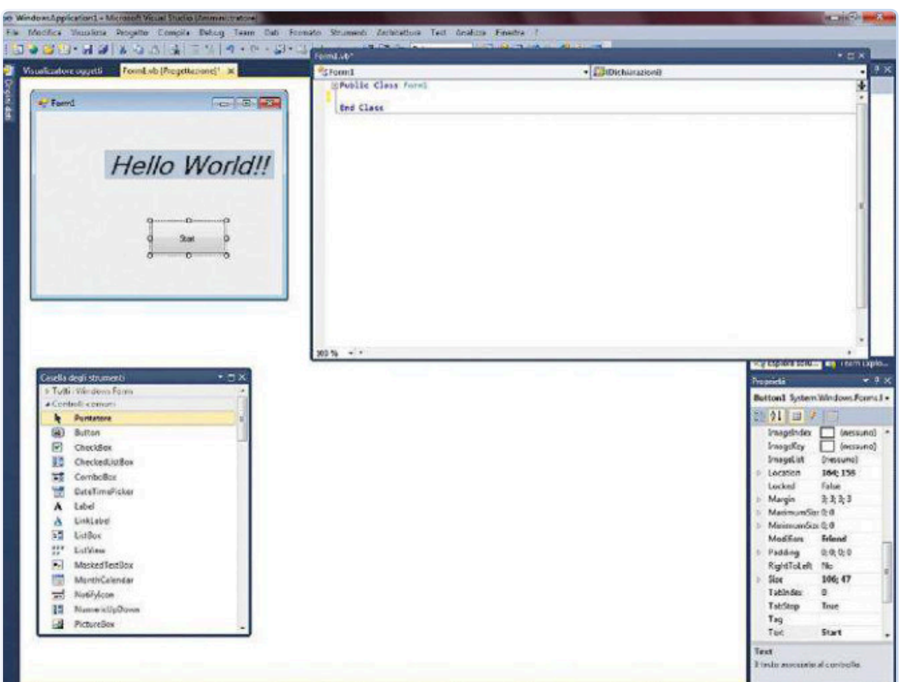
La **ridondanza del codice** è la necessità di riscrivere più volte le medesime istruzioni o istruzioni simili. La situazione opposta, più vantaggiosa per il programmatore, viene detta **riuso del codice**.

La programmazione a oggetti è maggiormente intuitiva per programmatori non esperti e permette una rappresentazione semplice della struttura concettuale (come risulta dallo schema sottostante).



Lo schema sopra riportato rappresenta, mediante una simbologia frequentemente utilizzata, le classi come blocchi comprendenti due parti: gli attributi e i metodi. Nell'esempio potenza, cilindrata e coppia motrice costituiscono gli attributi dell'automobile; tra i metodi è indicato un procedimento che, in base alla potenza rilevata dagli attributi, stima il consumo dell'auto.

I programmi IDE in ambiente Windows sono particolarmente adatti alla programmazione a oggetti. Un'interfaccia visuale permette di costruire il modello grafico del programma e associare a ciascun oggetto inserito il relativo codice. Le istruzioni inserite possono essere abbinate alla classe di appartenenza degli oggetti e ai relativi metodi.
Per esempio, nella figura di pagina successiva è rappresentata un'applicazione Windows costruita nell'ambiente Microsoft Visual Studio. Sono stati collocati, nella finestra in alto a sinistra, due oggetti: un pulsante e una scritta "Hello world!". Ogni oggetto inserito appartiene a una specifica classe (per esempio, i pulsanti e le etichette). Gli eventi e le proprietà degli oggetti (per esempio, l'evento di click del mouse sul pulsante, la modifica del colore della scritta nell'etichetta) sono programmabili, nella finestra in alto a destra, attraverso la codifica di opportuni metodi



o funzioni associate ai singoli oggetti (il pulsante <X>) o a classi di oggetti (la classe dei pulsanti).

La modularità e la semplicità con cui tali interfacce permettono di creare modelli, anche estremamente complessi, rende la programmazione a oggetti e i relativi software particolarmente adatti alla risoluzione di problematiche avanzate.

ESERCITAZIONE GUIDATA 1

Flow chart

Riprodurre, con l'uso del word processor Microsoft Word 2007, il flow chart riportato in fig. 1.

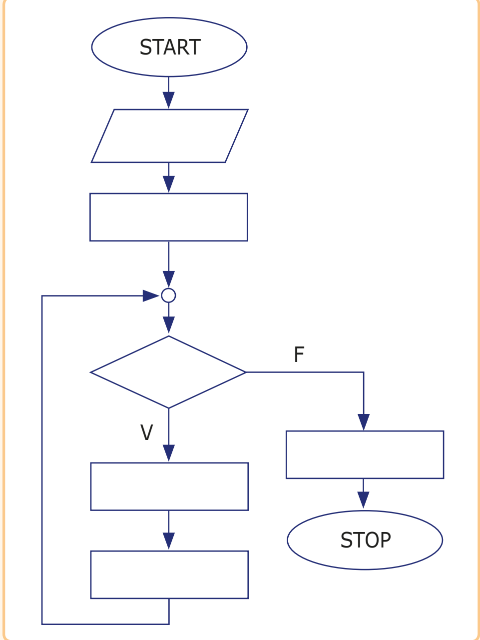


Fig. 1

Al significato dei simboli usati e dello svolgimento dei flow chart è dedicata parte dell'Unità seguente.

Avviata l'applicazione Microsoft Word 2007, al nuovo file è assegnato il nome FLOW CHART.
Nella prima riga si digita il titolo Costruzione di un flow chart, cui si assegna lo stile Titolo scegliendo il pulsante in <Home-Stili>.

ESERCITAZIONE GUIDATA 1

Si inizia la costruzione del flow chart usando Inserisci-Illustrazioni-Forme, cioè la stessa procedura da usare per eseguire disegni.

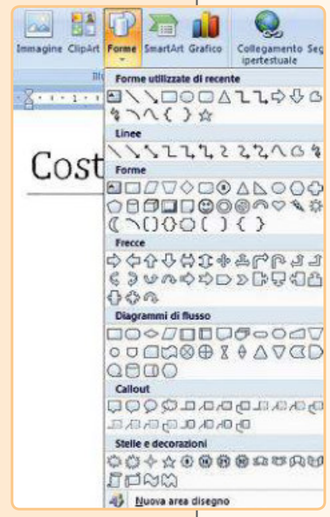


Fig. 2

Nel lungo menu di forme disponibili (fig. 2) inizialmente si sceglie la voce in fondo Nuova area disegno, che crea sul foglio una delimitazione entro cui sarà collocato il flow chart.

L'area di disegno (fig. 3) viene a costituire una cornice entro cui possono essere disegnate più forme o inserite immagini; sarà possibile poi formattare il bordo e lo sfondo dell'area, variarne le dimensioni e la collocazione.

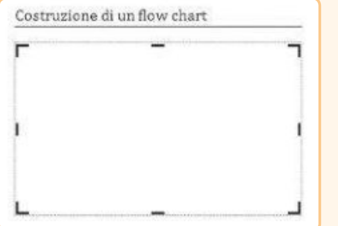
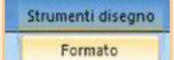


Fig. 3

Converrà intanto, prevedendo che il flow chart abbia uno sviluppo verticale, allungare verso il basso il bordo dell'area; si porta il cursore sul trattino al centro in basso e, tenendo premuto il cursore quando ha assunto la forma di una piccola T, si trascina verso il basso.
L'operazione di ampliamento può anche essere eseguita più tardi in base alla necessità di spazio.



Se i bordi dell'area diventano invisibili è sufficiente cliccare nello spazio relativo; si apre il menu Strumenti disegno-Formato che consente di effettuare tutte le operazioni principali.

Con Inserisci forme, cliccando sul pulsantino in basso a destra (fig. 4), si riapre la finestra delle forme di disegno disponibili; quelle utili per disegnare il flow chart sono nella sezione Diagrammi di flusso.

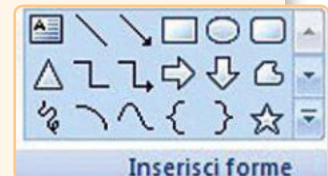


Fig. 4

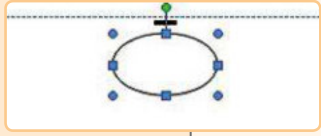


Fig. 5

Iniziamo con il cerchio (etichetta Connettore) per disegnare in realtà un'ellisse: dopo averlo selezionato, collochiamo il cursore sull'area di disegno, in alto al centro; tenendo premuto, delimitiamo un rettangolo in cui è inscritto l'ovale (fig. 5).

Facendo passare il mouse sulla figura, assume la forma di una croce: serve, puntando e trascinando, a spostare la forma.

Portando invece il mouse sui quadratini del contorno, esso diventa una freccetta che serve per allargare/stringere la figura; di conseguenza l'ellisse può avvicinarsi o allontanarsi sempre più dalla forma circolare.

È bene tenere presente che, se si agisce sui quadratini d'angolo, le dimensioni si modificano senza variare il rapporto larghezza/lunghezza.

ESERCITAZIONE GUIDATA 1

Poco sotto al cerchio si colloca un parallelogramma (etichetta Dati) con la stessa tecnica e ancora sotto inseriamo un rettangolo (etichetta Elaborazione) (fig. 6).

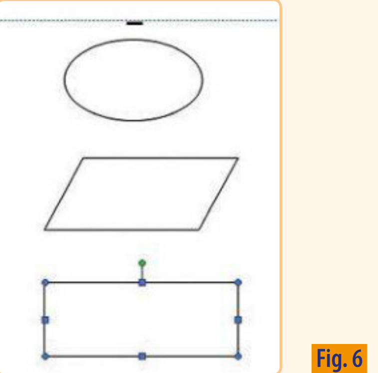


Fig. 6

Prima di proseguire con altre forme effettuiamo i collegamenti tra quelle già disegnate, usando una linea con la freccia, disponibile nella sezione Linee del menu Inserisci forme; le varie forme (rettangolo, parallelogramma ecc.) presentano, quando si avvicina il cursore, dei quadratini blu a cui vanno automaticamente a collegarsi le estremità della linea. Se la direzione finale della linea non è verticale, si individua la figura e la si sposta orizzontalmente in modo opportuno (fig. 7).

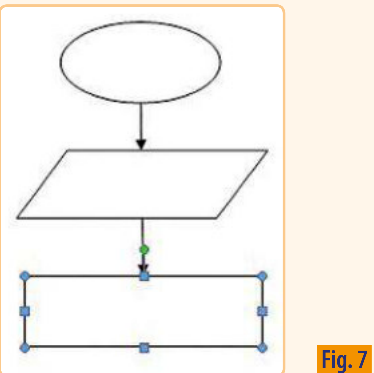


Fig. 7

Allo stesso modo si costruisce la parte restante del flow chart, usando di nuovo il cerchietto, poi il rombo, tre rettangoli e infine l'ellisse. Le linee spezzate sono realizzate usando tratti di linea senza freccia (fig. 8).

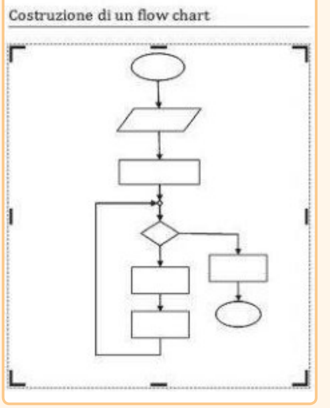


Fig. 8

ESERCITAZIONE GUIDATA 1

Nei flow chart sono inserite anche scritte che chiariscono il ruolo di ciascun blocco; in questo caso ne introduciamo due soltanto:

- INIZIO (o START) nell'ellisse iniziale;
- FINE (o END) nell'ellisse finale.

Il modo più facile consiste nel disegnare all'interno della figura una *casella di testo*, cioè un riquadro entro cui si può inserire una scritta, usando l'icona con una lettera A all'interno di un rettangolo (fig. 9).

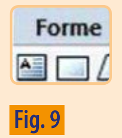


Fig. 9

Portato il cursore all'interno dell'ellisse in alto, si disegna un rettangolo entro cui si digita INIZIO o START; occorre disporre la casella in modo che stia tutta all'interno dell'ellisse.

Evidenziandone il bordo, compare il cursore a croce e la casella può essere spostata. Inoltre se si clicca con il tasto destro del mouse sul bordo si apre un menu rapido (fig. 10); scegliendo Formato casella di testo si possono modificare le caratteristiche della casella: alla voce Linea-Colore si sceglie il bianco rendendo così invisibile il bordo della casella, mentre resta evidenziata la scritta.

Fig. 10

Un'operazione simile di formattazione può essere compiuta sull'intera area del disegno: si clicca sul suo contorno con il tasto destro e si imposta un bordo di colore nero, mentre alla voce Riempimento-Colore si sceglie un grigio leggero.

Dopo l'OK il risultato finale è quello proposto in fig. 11.

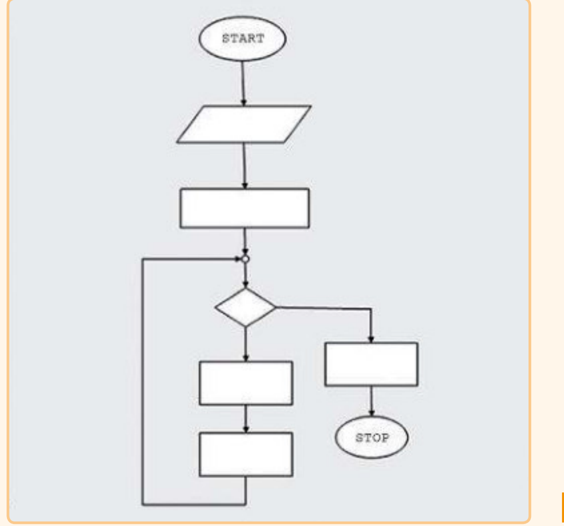
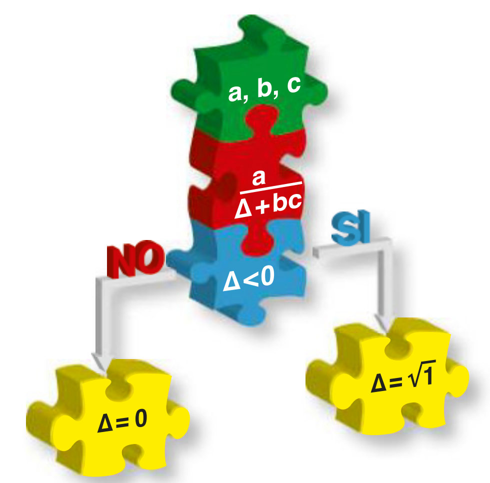


Fig. 11

Progettare algoritmi per il problem solving



Nella precedente Unità abbiamo introdotto i concetti preliminari della programmazione e abbiamo descritto i principali passi da compiere per affrontare un problema dal punto di vista informatico. In questa Unità ci soffermeremo sulla progettazione di algoritmi.

2.1 La progettazione di un algoritmo

La **programmazione** serve ad automatizzare processi di risoluzione dei problemi altrimenti demandati a soluzioni manuali. L'uso di specifici linguaggi di programmazione per l'interazione con il computer permette di tradurre le soluzioni adottate in una codifica che la macchina è in grado di interpretare.

In questa Unità ci soffermeremo sulla fase che precede la codifica, la **progettazione**, introducendo due formalismi, uno grafico e uno testuale. Non è ragionevole affrontare un problema direttamente mediante uno specifico linguaggio di programmazione per i seguenti motivi:

- l'algoritmo che viene proposto come soluzione deve avere validità generale, che prescinde dal linguaggio di codifica scelto;
- i vincoli del linguaggio adottato non dovrebbero precludere la scelta della soluzione da adottare. Al contrario, partendo da una proposta consolidata di algoritmo risolutivo, l'analista sceglierà, in un secondo tempo, il linguaggio di programmazione che risulta più adatto alle sue esigenze;
- la soluzione descritta mediante un linguaggio di programmazione è interpretabile pienamente solo da analisti esperti nel linguaggio.

Interprete: programma che traduce le istruzioni una per volta per eseguirle senza bisogno di ricompilare tutto il codice ad ogni modifica.

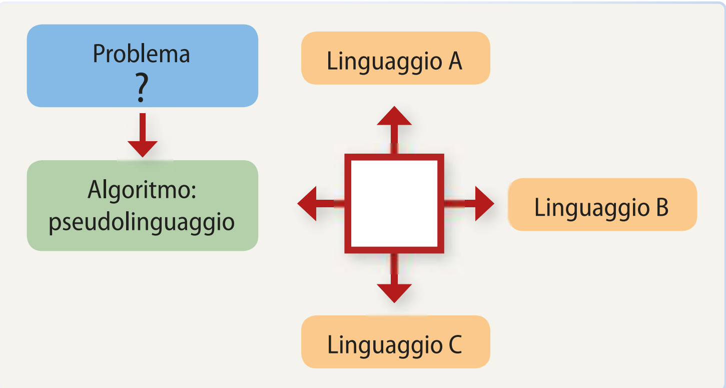
Compilatore: programma che rende eseguibile direttamente un codice sorgente traducendolo da una forma testuale alla forma binaria.

Allo scopo di semplificare la procedura di progettazione di algoritmi e renderla indipendente dalla fase di codifica vera e propria (implementazione) sono stati introdotti gli *pseudolinguaggi*.

Lo **pseudolinguaggio** è un linguaggio artificiale, facilmente interpretabile ed analizzabile senza l'ausilio di specifici **interpreti** o **compilatori**, avente lo scopo di descrivere procedure algoritmiche con valenza generale, senza addentrarsi nelle specifiche tecniche dei singoli linguaggi di programmazione.

Gli pseudolinguaggi descrivono un algoritmo dal punto di vista concettuale.

© RCS LIBRI EDUCATION SPA



Riprendendo il concetto di **livello di un linguaggio di programmazione** introdotto nella precedente Unità, si può affermare che lo pseudolinguaggio rappresenta un linguaggio per la programmazione ad un livello ancora superiore rispetto ai linguaggi ad alto livello, in quanto traccia la linea di soluzione nel modo più generale possibile.

In questa Unità analizzeremo le caratteristiche principali di due differenti tipologie di pseudolinguaggi: i **flow chart** (detti anche **diagrammi di flusso**) e lo **pseudocodice** (detto anche **linguaggio di progetto**): entrambi formalismi utili per rappresentare algoritmi.

Livello di un linguaggio di programmazione: livello che, in linea teorica, indica il numero di istruzioni in linguaggio macchina a cui corrispondono mediamente le istruzioni del linguaggio.

Flow chart

I flow chart costituiscono un metodo grafico di rappresentazione e descrizione di un algoritmo. Essi combinano forme grafiche (per esempio, rettangoli e rombi) con descrizioni testuali; il tipo di figura geometrica usata indica la tipologia di istruzione impartita, mentre il testo in essa racchiuso dà informazioni più dettagliate sul passo algoritmico descritto.

Un flow chart è quindi una *mappa concettuale* che rappresenta mediante un diagramma la sequenza logica di blocchi che il programma deve eseguire.

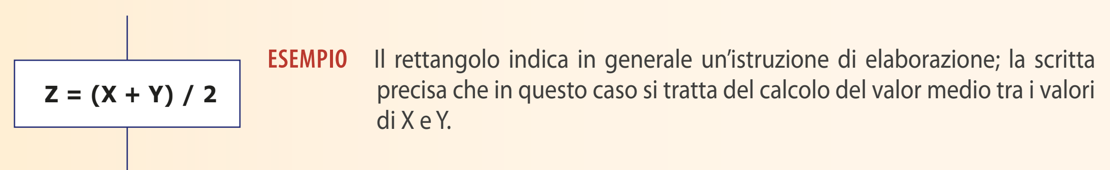
La **mappa concettuale** è uno strumento grafico utile per rappresentare informazioni o concetti.

ESEMPIO Spesso si usano le mappe concettuali a scopo didattico per riassumere e far comprendere meglio i concetti fondamentali relativi ad uno specifico argomento. Il flow chart è invece uno strumento utilizzato a fini operativi per supportare la costruzione di progetti complessi, come i programmi informatici utili in ambito aziendale.

Esempi
Fai un esempio di mappa concettuale che usi per lo studio di una materia a tua scelta.

Ciascun blocco racchiude una duplice informazione:

1. l'informazione di alto livello sulla classe a cui appartiene, deducibile dalla forma della sagoma rappresentata;
2. l'informazione più dettagliata sulla specifica indicata nell'istruzione.



ESEMPIO Il rettangolo indica in generale un'istruzione di elaborazione; la scritta precisa che in questo caso si tratta del calcolo del valor medio tra i valori di X e Y.

Le istruzioni di un generico algoritmo possono non essere eseguite in modo lineare dall'inizio alla fine in un'unica passata. Possono essere presenti *cicli* del tipo **RIPETI N VOLTE...** o **RIPETI FINO A QUANDO...** oppure *vincoli condizionali* (**SE... ALLORA... ALTRIMENTI**), che modificano l'ordine e il numero di volte con cui le istruzioni vanno eseguite.

Il **flusso** d'esecuzione di un algoritmo è quindi variabile e dipende dai dati in ingresso forniti dall'utente. La sequenza con la quale le istruzioni devono essere eseguite è rappresentata da frecce di collegamento tra i vari blocchi (*sagome*).

Il **flusso dell'algoritmo** è una possibile sequenza di istruzioni che viene lanciata durante l'esecuzione di un algoritmo, in seguito all'introduzione di specifici dati di ingresso.

ESEMPIO In un programma che deve calcolare il rapporto tra due numeri scelti in ingresso, X e Y, deve essere inserita una condizione d'uscita per cui l'istruzione di divisione non viene eseguita se il divisore è uguale a zero (la divisione perde significato e risulta indefinita).

Di conseguenza se l'utente inserisce 0 come valore d'ingresso della variabile Y, il programma segnala all'utente l'avvenuto errore e alcune delle istruzioni previste dall'algoritmo non sono eseguite.

Quello sopra riportato è un esempio di **analisi del flusso dell'algoritmo**: si individuano le possibili esecuzioni reali dell'algoritmo per controllare quale sequenza di istruzioni viene effettivamente eseguita in funzione dei dati in ingresso.

In base al legame tra i dati in ingresso e in uscita, gli algoritmi si dividono in **deterministici** e **non deterministici**.

L'**algoritmo deterministico** è un algoritmo in cui, in seguito all'introduzione della medesima sequenza di dati in ingresso (*input*), viene generata sempre la medesima sequenza di dati in uscita (*output*).

L'**algoritmo non deterministico** è un algoritmo in cui, in seguito all'introduzione della medesima sequenza di dati in ingresso (*input*), possono essere generate sequenze differenti di dati in uscita (*output*).

ESEMPIO Un algoritmo che simula il lancio di un dado può prevedere la generazione di un numero intero casuale da 1 a 6 e la stampa del valore in uscita. In questo caso supponiamo che l'utente possa indicare in ingresso il numero di lanci e che in uscita venga prodotto, oltre alla sequenza di numeri ottenuti, anche quante volte è uscito il numero 6.
L'algoritmo in questione è di tipo *non deterministico*, perché con un arbitrario numero di lanci richiesto in ingresso è possibile ottenere risultati completamente differenti.

Al contrario, consideriamo nuovamente il problema della divisione di due numeri X e Y; se il programma produce in uscita il risultato della divisione, l'algoritmo risulta *deterministico* perché fissati i due numeri in ingresso, X e Y, si può ottenere un unico risultato.

È da notare che in un algoritmo non deterministico il flusso dell'algoritmo può cambiare a fronte della medesima sequenza d'ingresso.

ESEMPIO Supponiamo che nell'algoritmo sul lancio dei dadi dell'esempio precedente sia inserita la condizione aggiuntiva per cui se il numero 6 non esce dopo un determinato numero di lanci, il risultato non viene prodotto. L'avverarsi di questa condizione determina la modifica del flusso dell'algoritmo. La probabilità che questo evento accada dipende dal numero di lanci specificato dall'utente: aumentando il numero di lanci a diposizione diminuisce la probabilità che il 6 non esca nemmeno una volta.

La possibilità di rappresentare in modo grafico la sequenza di istruzioni di un algoritmo rende più chiara e intuitiva l'analisi dei flussi, mentre la lettura di un

> **Listato**: termine utilizzato per denotare una sequenza di linee di codice (istruzioni) in forma testuale.

listato di programma spesso non permette una visione d'insieme schematica della struttura dell'algoritmo.

Gli svantaggi della rappresentazione con i flow chart possono essere riassunti nei seguenti punti:
- la rappresentazione grafica è poco compatta e quindi lunga da scrivere;
- l'insieme delle classi di istruzioni corrispondenti alle tipologie di blocchi è limitato.

La costruzione di un diagramma di flusso richiede tempo; se la complessità del problema aumenta, la rappresentazione cartacea diventa difficoltosa e la costruzione su computer viene saltata dagli analisti più esperti per risparmiare tempo. Inoltre, la complessità dei problemi richiede spesso l'uso di istruzioni non classificabili in modo rigoroso in una delle categorie attualmente in uso nell'ambito della progettazione di flow chart. D'altro canto, l'uso di una notazione grafica non largamente interpretabile renderebbe lo sforzo di rappresentazione poco produttivo. Per questo motivo viene spesso utilizzato uno pseudolinguaggio in formato esclusivamente testuale, che garantisce maggiore flessibilità e rapidità di stesura: lo **pseudocodice** (o linguaggio di progetto).

Pseudocodice

> **Linker**: programma che mette insieme diversi sottoprogrammi compilati separatamente per ottenere un programma operativo completo.

Lo pseudocodice è la tipologia di pseudolinguaggio più vicina ai linguaggi di programmazione veri e propri; come i linguaggi di codifica, prevede una sequenza di istruzioni testuali che vanno a comporre un listato.
Ciò che lo distingue da un linguaggio vero e proprio è il fatto che non deve sottostare alle specifiche di un determinato compilatore, **linker** o interprete per quanto riguarda la sintassi e il lessico usati.

ESEMPIO Per costruire un ciclo la cui esecuzione è condizionata ad un evento prefissato esistono, a seconda dei linguaggi di programmazione, sintassi diverse:

```
REPEAT {
...
sequenza di istruzioni da eseguire
...
} UNTIL (condizione di uscita)
```

```
DO {
...
sequenza di istruzioni da eseguire
...
} WHILE (condizione di permanenza nel ciclo)
```

> **Repeat until**: "ripeti fino a quando".
>
> **Do while**: "esegui mentre".

L'uso di un particolare linguaggio vincola l'esperto all'impiego di una specifica forma in quanto il compilatore (o l'interprete) non è in grado di tradurre in linguaggio macchina istruzioni che differiscono dalla sintassi specifica del linguaggio. Nella scrittura di pseudocodice, invece, non sono imposti vincoli sul formato del comando da impartire: in altre parole, si può usare una o l'altra delle precedenti forme verbali oppure si può scrivere:

```
RIPETI FINCHÉ (condizione di uscita)
INIZIO:
...
sequenza di istruzioni da eseguire
...
FINE
```

Sarà l'analista a scegliere di adottare una forma verbale piuttosto di un'altra perché la ritiene maggiormente intuitiva o più vicina alle convenzioni in uso.

ESEMPIO Gli autori di articoli e pubblicazioni legati alla medesima rivista o al medesimo settore adottano un proprio linguaggio comune. La scelta del lessico, della sintassi e della semantica è nata dall'esperienza e dalla pratica comune che hanno contribuito a convergere verso una notazione condivisa.

Gli unici vincoli che si impongono a un linguaggio di progetto sono:
- **interpretabilità** e **leggibilità**, per cui il linguaggio deve essere il più vicino possibile al linguaggio naturale e lontano dal linguaggio macchina;
- **indipendenza dai vari paradigmi di programmazione**, nonostante la tendenza di ogni programmatore ad avvicinarsi al linguaggio di programmazione preferito o maggiormente conosciuto.

La **comprensibilità del codice** è una proprietà che misura quanto una codifica sia facilmente interpretabile e riapplicabile anche da utenti non esperti del settore. Esiste uno stretto legame tra **comprensibilità** e **riusabilità**.

L'esperienza dei programmatori ha portato ad un uso diffuso di parole chiave, che introdurremo in parallelo all'uso dei flow chart.

Le parole-chiave più comunemente usate negli pseudocodici corrispondono alle macro-categorie e micro-categorie di istruzioni rappresentabili mediante flow chart.

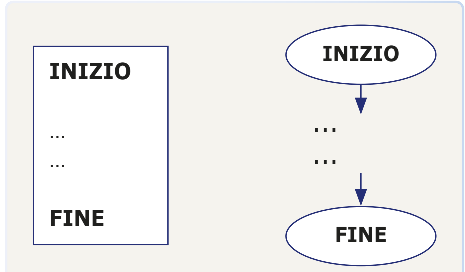
Vedremo per ciascuna classe di istruzioni:
- il significato generale;
- i termini maggiormente usati nella stesura di pseudocodici;
- la notazione grafica usata per i flow chart (quando disponibile).

2.2 Le notazioni in uso negli pseudolinguaggi

Analizziamo ora i costrutti fondamentali utilizzati nella descrizione di un algoritmo e le relative rappresentazioni nel flow chart e nello pseudocodice.

Inizio e Fine

Un algoritmo è sempre caratterizzato da un punto di inizio e da un punto di fine:



INIZIO (*Start*) e **FINE** (*End*) sono le parole chiave che nello pseudocodice individuano l'avvio e il termine del flusso.
Nei flow chart i blocchi **INIZIO** e **FINE** si rappresentano mediante un **ovale** contenente la rispettiva parola chiave.

Nello schema sono rappresentati affiancati **INIZIO** e **FINE** in uno pseudocodice e relativo flow chart.

Variabili: definizione e assegnazioni

Uno degli *elementi base* dei linguaggi di programmazione, come abbiamo visto nell'Unità precedente, è la *variabile* che serve a memorizzare ed elaborare dati.

Le **variabili** sono dati a cui si associa un'area riservata della memoria centrale. Il modo con cui sono memorizzate dipende dal tipo di dato che si vuole immagazzinare (per esempio numero intero, numero reale, carattere ecc.).

Le principali operazioni che coinvolgono una variabile sono:

1. definizione;
2. assegnazione e inizializzazione.

1. La **definizione di una variabile** consiste nell'allocazione dell'area di memoria dedicata, assegnando ad essa un nome e un tipo. Per esempio, la definizione della variabile Età riserva un'area di memoria per memorizzare un valore intero, a cui intendiamo riferire genericamente l'età di una persona.

   Nella elaborazione di un algoritmo il numero di variabili da definire dipende dalla complessità del problema e dalle esigenze del programmatore; una buona regola di programmazione è definire il numero minimo di variabili necessario per raggiungere il risultato.

   Il tipo di dato assegnato alla variabile, specifica il formato di memorizzazione (per esempio numerico, testo ecc.). Tale informazione verrà presa in considerazione solo a livello di codifica, in quanto **gli pseudolinguaggi sono indipendenti dal linguaggio di programmazione prescelto**.

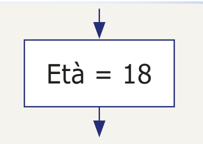
2. L'**assegnazione di una variabile** consiste nell'attribuire un valore, nell'area di memoria preposta, coerente con il tipo di dato specificato.

   La prima **assegnazione** che viene effettuata dopo la definizione della variabile viene detta **inizializzazione**. L'inizializzazione di una variabile è una particolare assegnazione che viene eseguita nella parte iniziale dello pseudocodice o comunque prima che la variabile possa subire riassegnazioni. Inizializzare le variabili è importante per evitare incongruenze dovute a valori inattesi memorizzati, non volontariamente, nell'area di memoria assegnata.

   Nello pseudocodice l'assegnazione e l'inizializzazione sono indicate con il nome della variabile seguita dal segno di uguale (=) e dal valore assegnato.

ESEMPIO Di seguito sono elencati tre esempi di assegnazione alle variabili: **Età**, **Nome**, **Data**. Il contenuto da assegnare alla variabile **Nome** è di tipo alfanumerico e nella maggior parte dei linguaggi viene incluso tra apici "...".

Età = 18
Nome = "Giovanni"
Data = 1/4/2015.



Nel flow chart le assegnazioni appartengono alla classe delle elaborazioni, indicate con un rettangolo al cui interno sono riportati il nome della variabile e il valore.

Nonostante siano previste nei linguaggi di programmazione moderni tipologie di variabili più complesse, soffermiamoci sul tipo più semplice, in cui la **variabile può assumere un unico valore**. In questo caso, se durante il corso del flusso dell'algoritmo avviene una successiva assegnazione alla medesima variabile con un valore diverso, questo andrà a sostituire quello precedente.

UTILE A SAPERSI

Il significato del segno = in informatica

L'uso del segno = non deve trarre in inganno: in informatica, a differenza della matematica, può indicare un'assegnazione. Per esempio, se consideriamo la seguente espressione:

Età = Minimo

essa significa che alla variabile **Età** è assegnato il valore attuale della variabile **Minimo**.

Se la successiva assegnazione riguarda una variabile numerica, essa può consistere in un incremento.

ESEMPIO L'assegnazione

Età = Età + 1

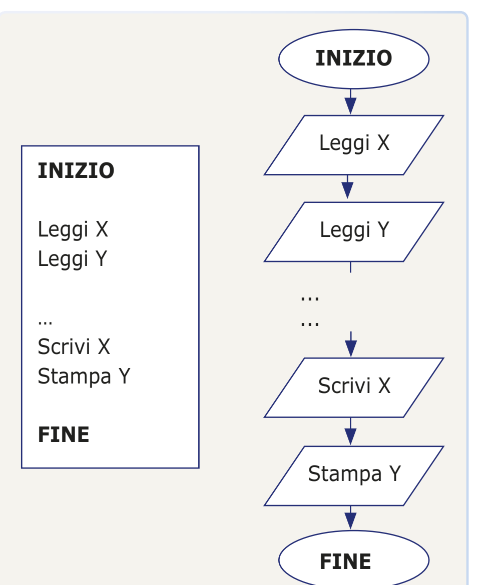
incrementa di una unità il valore della variabile **Età**; al precedente valore 18 viene sostituito il valore 19. A destra dell'uguale si riporta il nome della variabile come riferimento al suo valore prima della modifica (per esempio, 18). A sinistra dell'uguale si pone invece la variabile in cui verrà memorizzato il risultato della somma algebrica 18 + 1 = 19.
In questo caso particolare, la lettura e la memorizzazione coinvolgono la medesima variabile (**Età**) e, dunque, il suo valore finale risulterà incrementato di uno.

Se, per esigenze legate allo svilupparsi dell'algoritmo, il valore temporaneo di una variabile deve essere conservato, diventa necessario introdurre una seconda variabile.

ESEMPIO Prima di incrementare la variabile **Data**, il suo valore, che rappresenta una scadenza, viene assegnato alla variabile **Data_Scadenza**:

Data = 1/5/2012
…
Data_Scadenza = Data
Data = Data + 30
…

Il risultato è:
Data = 31/05/2012



Ingressi e uscite

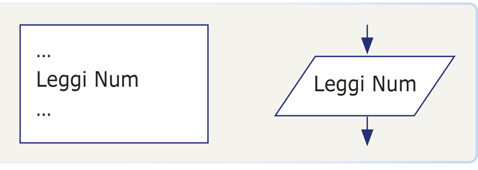
L'analisi del problema impone di determinare i dati di ingresso e di uscita che saranno processati dall'algoritmo. Per indicare la lettura di un dato di ingresso (input) o la restituzione di un dato di uscita (output) nei flow chart si usa un blocco a forma di parallelepipedo, come rappresentato nello schema.

Nello pseudocodice si indica solitamente la descrizione dell'operazione eseguita, che può essere per esempio, **Leggi** (*Read*), **Scrivi** (*Write*) o **Stampa** (*Print*).

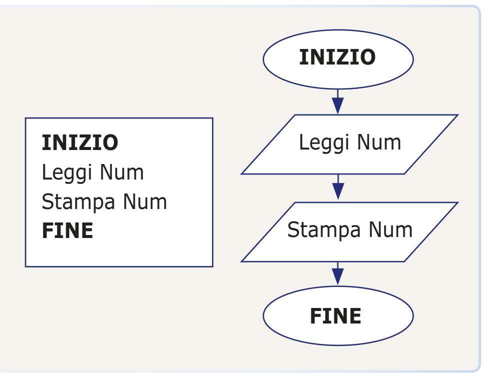
Una tipica operazione di lettura può ricevere un dato numerico inserito da tastiera e memorizzarlo all'interno di una variabile denominata Num. Il corrispondente blocco di lettura sarà rappresentato come nello schema di pagina successiva.

Memoria di massa esterna: termine che può indicare un CD-ROM, un DVD o una chiavetta USB.

Comuni operazioni di output sono invece la stampa del valore di una variabile a video o la scrittura su un file registrato su disco o su una **memoria di massa esterna**.



Se, per esempio, si vuole leggere un valore numerico da tastiera e poi stamparlo a video, le traduzioni rispettive nel flow chart e nello pseudocodice sono quelle rappresentate negli schemi a destra.



Elaborazione

I dati sono rielaborati dall'algoritmo mediante opportune operazioni di calcolo. Per ottenere il risultato, si usano spesso delle **variabili di appoggio**, in aggiunta a quelle di ingresso e di uscita, a cui vengono assegnati valori provvisori, utili per il raggiungimento del risultato finale.

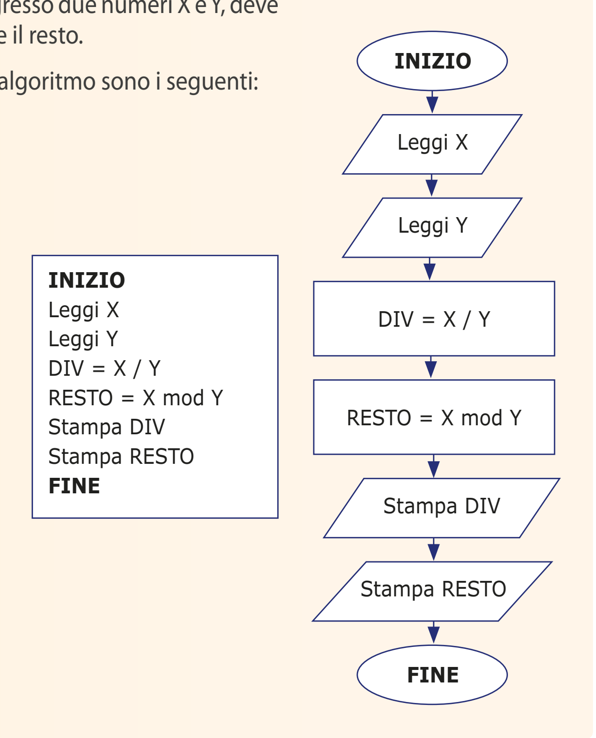
Nel flow chart l'elaborazione si rappresenta mediante una sagoma rettangolare, contenente l'operazione di calcolo, di assegnazione o di aggiornamento di variabili. Nello pseudocodice si riporta invece direttamente la descrizione dell'operazione senza l'aggiunta di ulteriori parole chiave.

ESEMPIO Consideriamo un algoritmo che, dati in ingresso due numeri X e Y, deve calcolare il risultato della divisione intera e il resto.

Lo pseudocodice e il flow chart relativi all'algoritmo sono i seguenti:

Nell'esempio di algoritmo si leggono i valori di ingresso X e Y e si compiono due operazioni:

- la divisione intera (indicata con la barra /), che fornisce il risultato della divisione senza resto (per esempio, 5 / 2 = 2)
- il modulo (indicata con **mod**) che fornisce il resto della divisione intera (per esempio, 5 mod 2 = 1).



DIV e **RESTO** sono due variabili di appoggio, il cui valore verrà restituito in output. I blocchi di calcolo e assegnazione di valore alle variabili **DIV** e **RESTO** sono esempi di blocchi di elaborazione.

Espressioni condizionali e indentazione

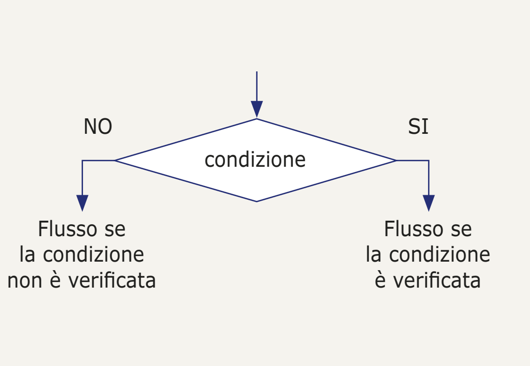
Il flusso dell'algoritmo può presentare ramificazioni; una o più condizioni (per esempio, sul valore di una determinata variabile) determinano quale ramo il flusso deve seguire; esse quindi influiscono sull'output dell'algoritmo.

ESEMPIO L'algoritmo che gestisce la fruizione dei servizi automatici di distribuzione dei tabacchi deve, per legge, verificare che la persona richiedente sia maggiorenne. Quindi, in seguito a una verifica mediante lettura magnetica della tessera sanitaria, l'algoritmo deve prendere una decisione. La condizione che determina l'erogazione o meno del prodotto è il valore della data di nascita associata al richiedente. Qualora dalla data di nascita sia desumibile che il cliente è maggiorenne, l'erogazione viene realizzata; altrimenti viene segnalato il problema e bloccata la fornitura.

L'espressione condizionale si esprime con le parole chiave:
SE... ALLORA... ALTRIMENTI (*If... Then... Else*).

A seguito della parola **SE** viene indicata la condizione principale da verificare. Se la condizione risulta verificata, l'algoritmo prosegue seguendo il flusso di istruzioni associato alla parola chiave **ALLORA**; al contrario, se la condizione non è verificata l'algoritmo prosegue seguendo le istruzioni indicate dalla parola chiave **ALTRIMENTI**.
Nella maggior parte dei linguaggi la parola **ALTRIMENTI** non è obbligatoria; in sua assenza si intende che, se non è verificata la condizione iniziale, l'algoritmo segue il suo flusso principale.

```
...
SE (condizione)
ALLORA
   Istruzioni da eseguire
   se la condizione
   è verificata
ALTRIMENTI
   Istruzioni da eseguire
   se la condizione
   non è verificata
...
```

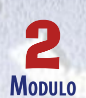


La corrispondente rappresentazione mediante flow chart prevede l'uso di un blocco romboidale in cui viene scritta la condizione. I possibili valori assunti dalla condizione (per esempio, Sì/No o Vero/Falso) corrispondono a differenti ramificazioni che partono dal rombo e vanno a unirsi ai blocchi che devono succedere nell'ordine previsto dall'algoritmo stesso.

Il flusso dell'algoritmo in esecuzione cambia a seconda delle variabili di input; in relazione ai dati immessi, la condizione potrà essere verificata o meno. Dalla condizione dipende quindi quali istruzioni successive verranno eseguite.

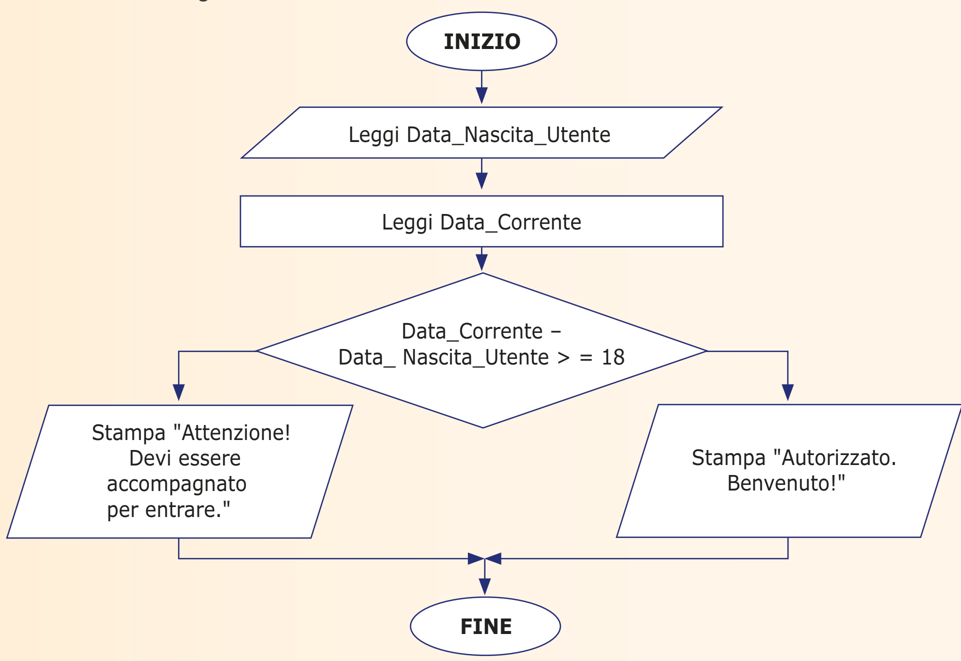
ESEMPIO Supponiamo di voler progettare un algoritmo per verificare l'età di un partecipante ad un torneo. Il programma deve richiedere al partecipante la data di nascita e, qualora non sia maggiorenne, segnalare che deve essere accompagnato da un genitore.
Quella seguente è una possibile soluzione.

```
INIZIO
Leggi Data_Nascita_Utente
Leggi Data_Corrente
SE (Data_Corrente – Data_ Nascita_Utente > = 18)
ALLORA
        Stampa "Autorizzato. Benvenuto!"
ALTRIMENTI
        Stampa "Attenzione! Devi essere accompagnato per entrare."
FINE
```



La condizione descritta dal listato è la seguente:
SE la differenza tra la data corrente e la data di nascita è maggiore o uguale a 18
ALLORA stampa il messaggio di benvenuto
ALTRIMENTI stampa il messaggio di richiesta di accompagnamento.

Il flow chart è il seguente.



Implementare: scrivere il codice relativo a un algoritmo in uno specifico linguaggio.

Lo pseudolinguaggio richiede all'utente in ingresso la data di nascita e stampa, in uscita, un messaggio di benvenuto o una richiesta di accompagnamento. La data corrente viene restituita da una funzione Leggi Data_Corrente, la cui specificazione non è d'interesse nel contesto di uno pseudolinguaggio; andrà invece codificata quando si **implementerà** l'algoritmo in uno specifico linguaggio di programmazione.

La condizione verifica lo scarto tra la data attuale e quella di nascita; nello pseudolinguaggio non ci occupiamo del tipo di dato: i formati delle variabili utilizzate (Data, Numerico ecc.) andranno gestiti in modo opportuno solo durante la fase di codifica.

Per organizzare meglio lo pseudocodice dal punto di vista grafico e facilitarne la lettura e la revisione, si utilizza la tecnica dell'**indentazione**.

L'**indentazione** è una tecnica di scrittura che consiste, in caso di condizioni o cicli, nel formattare il codice mediante opportune tabulazioni, al fine di separare graficamente blocchi di istruzioni che logicamente nel flusso rientrano nella medesima condizione.

Nel caso dei blocchi condizionali, si inserisce una tabulazione per le istruzioni che rientrano nella condizione **ALLORA** e nella condizione **ALTRIMENTI** in modo da evidenziare graficamente la loro separazione dal resto del listato.

Le condizioni si caratterizzano per il numero di risultati possibili; sono denominate **condizioni booleane** quelle il cui risultato è VERO o FALSO.

ESEMPIO La condizione usata nell'esempio precedente sulla differenza tra le date è un caso di condizione booleana, che restituisce **VERO** se lo scarto è maggiore o uguale a 18, **FALSO** negli altri casi.

La condizione può assumere valori multipli: poiché nel costrutto **SE... ALLORA... ALTRIMENTI** si prevedono solo due casi possibili, per rappresentare condizioni più complesse si adottano più **condizioni innestate.**

Le **condizioni innestate** sono condizioni scritte una all'interno dell'altra per strutturare algoritmi più complessi che contengono condizioni multiple.

Esaminiamo la struttura di condizione innestata con l'aiuto di un esempio.

ESEMPIO Riprendendo il problema riguardante l'ingresso al torneo, supponiamo di voler considerare più fasce di età per gestire offerte promozionali. Ai minorenni accompagnati è proposta la tariffa scontata **MINI**; per persone tra i 18 e i 25 la tariffa di ingresso è quella **GIOVANI**, mentre oltre i 25 anni la tariffa è quella **PIENA**.
Una possibile soluzione è rappresentata dal seguente pseudocodice (fig. 1):

```
INIZIO
Leggi Data_Nascita_Utente
Tariffa = "PIENA"
Accompagnato = "NO"
Leggi Data_Corrente
SE (Data_Corrente – Data_ Nascita_Utente > = 18)
    ALLORA
        SE (Data_Corrente – Data_ Nascita_Utente < = 25)
            ALLORA
                Tariffa = GIOVANI
    ALTRIMENTI
        Tariffa = "MINI"
        Accompagnato = "SI"
        Stampa  Accompagnato
Stampa Tariffa
FINE
```

Fig. 1: Esempio di pseudocodice indentato.

```
INIZIO
Leggi Data_Nascita_Utente
Tariffa = PIENA
Accompagnato = NO
Leggi Data_Corrente
SE (Data_Corrente – Data_ Nascita_Utente > = 18)
ALLORA
SE (Data_Corrente – Data_ Nascita_Utente < = 25)
ALLORA
Tariffa = GIOVANI
ALTRIMENTI
Tariffa = MINI
Accompagnato = SI
Stampa Accompagnato
Stampa Tariffa
FINE
```

Fig. 2: Esempio di pseudocodice non indentato.

L'algoritmo deve considerare tre possibili casistiche di età: minore di 18 anni, tra i 18 e i 25 anni e maggiore di 25 anni.
Il valore della variabile **Tariffa** viene posto inizialmente pari a **PIENA** (inizializzazione), al fine di evitare incongruenze durante l'inizio del flusso algoritmico.
La prima condizione verifica se l'utente è maggiorenne o minorenne. In caso positivo, viene eseguita una seconda verifica, attraverso un secondo **SE** annidato nel precedente, con cui si verifica se l'età è

compresa tra 18 e 25. Se anche questa seconda condizione è soddisfatta allora il valore di **Tariffa** viene modificato in **GIOVANI**.
Nel caso in cui la prima condizione (> 18 anni) non fosse verificata la tariffa assegnata è **MINI** e viene notificato l'obbligo della presenza di un adulto accompagnatore.
Se la prima condizione (> 18 anni) è verificata mentre la seconda (≤ 25 anni) non lo è, allora l'utente ha più di 25 anni. In questo caso non è necessario riassegnare il valore della variabile **Tariffa** in quanto essa contiene già il valore corretto (**PIENA**) assegnatogli come valore iniziale.

La **Tariffa** assegnata viene infine stampata come valore in uscita.

Analizziamo ora l'indentazione usata nell'esempio. Alla prima condizione viene assegnato un primo rientro di tabulazione che distingue le istruzioni interne al primo **SE... ALLORA... ALTRIMENTI** da quelle iniziali. La seconda condizione aggiunge una seconda tabulazione per indicare il secondo condizionamento.

Confrontiamo lo pseudocodice indentato di fig. 1 con quello di fig. 2 che non lo è.

Nonostante entrambe le stesure siano formalmente corrette, la procedura di indentazione permette di aumentare notevolmente la leggibilità del codice, a vantaggio del programmatore.
Segue il corrispondente flow chart.

UTILE A SAPERSI

Sostegni extra

I programmi IDE supportano il programmatore per la scrittura del codice e offrono sostegni automatici o semi-automatici per l'indentazione. Per esempio, alla digitazione di un costrutto di condizionamento del tipo **SE... ALLORA...** propongono automaticamente la corretta indentazione rientrando di una tabulazione dopo l'a-capo.

INIZIO
Leggi Data_Nascita_Utente
Tariffa = PIENA
Accompagnato = NO
Leggi Data_Corrente
Data_Corrente – Data_ Nascita_Utente > = 18 ?
Falso
Vero
Tariffa = MINI
Accompagnato = SI
Stampa Accompagnato
Data_Corrente – Data_ Nascita_Utente < = 25 ?
Falso
Vero
Tariffa = GIOVANI
Stampa Tariffa
FINE

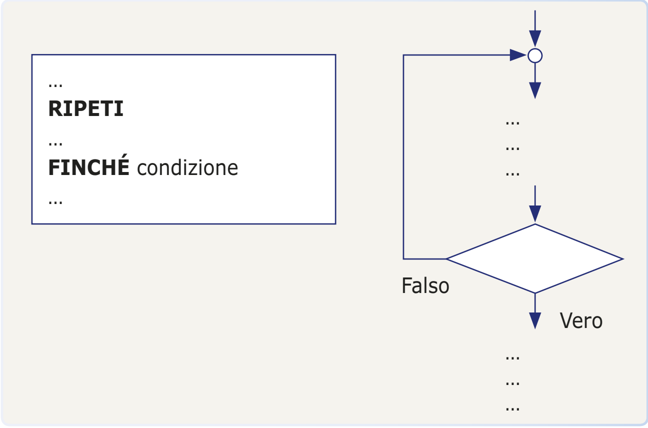
Cicli

I cicli rappresentano **iterazioni** sulla medesima porzione di codice o pseudocodice; vengono cioè ripetute istruzioni già precedentemente eseguite.

ESEMPIO Si vuole scrivere un algoritmo per calcolare la media dei voti di fine anno di uno studente di un istituto tecnico. Il programma non conosce a priori il numero di materie studiate durante l'anno; è opportuno, quindi, richiedere in ingresso, in modo iterativo, la denominazione di ciascuna materia e il voto finale ottenuto fino a quando l'utente non indica che l'elenco è terminato.
Un ciclo permette di ripetere la richiesta finché non è soddisfatta una condizione di uscita.

Le iterazioni sono rappresentabili mediante due simbologie simili, ma con significato parzialmente differente.

1. Nella prima modalità di rappresentazione del ciclo si indicano, mediante pseudocodice, le parole-chiave **RIPETI … FINCHÉ** (*Repeat … Until*).



Nello schema a sinistra è riportato lo pseudocodice e il corrispondente flow chart. La sequenza di istruzioni interne al ciclo si esegue almeno una volta. Al termine della prima esecuzione, un blocco condizionale permette di verificare se è possibile uscire dal programma oppure se è necessario iterare nuovamente il codice appena eseguito. In caso di esito negativo (condizione falsa), una freccia rimanda a un mini-blocco (costituito da un pallino vuoto) che rappresenta il punto di inserzione dell'algoritmo da cui riprendere il programma. Se la condizione risulta vera, il programma prosegue oltre.

Nello pseudocodice, il costrutto **FINCHÉ** condizione corrisponde al blocco condizionale riportato nel flow chart dove si verifica la condizione di uscita dal ciclo. In caso negativo, il flusso dell'algoritmo riprende dall'istruzione seguente il **RIPETI**, rappresentato graficamente nel flow chart mediante il pallino.

ESEMPIO Consideriamo ancora il calcolo della media dei voti finali di uno studente della scuola secondaria superiore. Una possibile soluzione mediante pseudocodice e flow chart è la seguente:

```
INIZIO
Continua = SI
Num_Materie = 0
Totale = 0
Media = 0
RIPETI
        Leggi Materia
        Leggi Voto
        Num_Materie = Num_Materie + 1
        Totale = Totale + Voto
        Leggi Continua // Inserimento Nuovo Voto?
FINCHÉ Continua = NO
Media = Totale / Num_Materie
Scrivi Media
FINE
```

All'inizio vengono inizializzate le variabili utilizzate nel programma:
– **Num_materie** memorizza il conteggio del numero di materie attualmente inserite;
– **Media** memorizza il calcolo della media dei voti;
– **Totale** contiene il risultato della somma dei voti inseriti;
– **Continua** è la variabile booleana (Vero/Falso; SI/NO) che indica se iterare nuovamente oppure no.

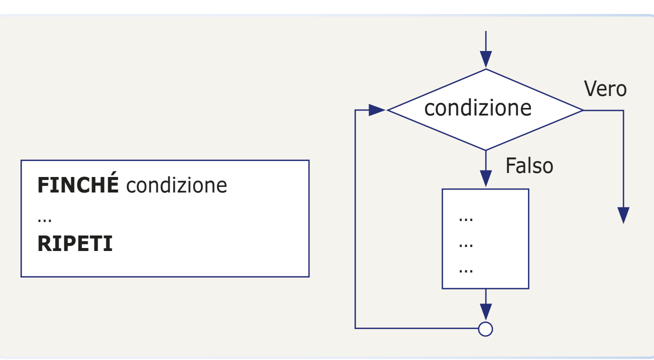
Le istruzioni interne al ciclo leggono in ingresso la materia e il relativo voto, aumentano di uno il conteggio del numero di materie e sommano il voto al totale dei voti correnti; infine richiedono all'utente se vuole inserire una nuova materia. Se l'utente digita **NO** il ciclo termina, altrimenti riprende una nuova iterazione e un nuovo inserimento di dati.

Il testo che segue il simbolo **//** è un **commento** dell'autore, utile per migliorare la chiarezza e la leggibilità del codice.

Al termine del ciclo viene calcolata la media e fornito il risultato come uscita dell'algoritmo.

2. Una seconda possibile formulazione del ciclo prevede la verifica della condizione all'inizio, prima dell'ingresso nel ciclo:
FINCHÉ condizione **... RIPETI** (*Until ... Repeat*)

Al contrario del caso precedente, questa formulazione implica che, se la condizione è soddisfatta, le istruzioni nel ciclo non vengono mai eseguite.




```
INIZIO
Continua = SI
Num_Materie = 0
Totale = 0
Media = 0
FINCHÉ CONTINUA = NO
        Leggi Materia
        Leggi Voto
        Num_Materie = Num_Materie + 1
        Totale = Totale + Voto
        Leggi Continua // Inserimento Nuovo Voto?
RIPETI
Media = Totale / Num_Materie
Scrivi Media
FINE
```

Il meccanismo di iterazione è analogo al precedente.

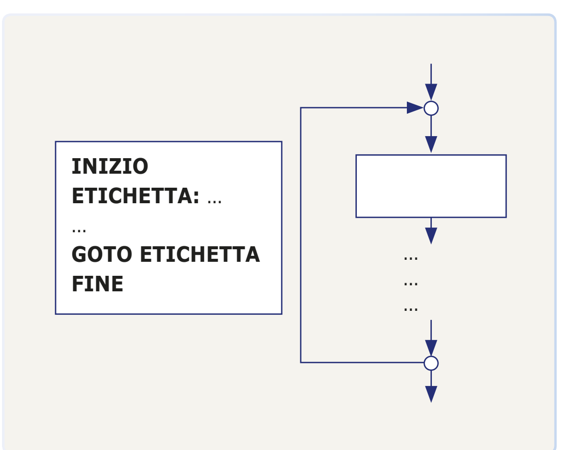
Il problema del calcolo della media dei voti può essere affrontato con un algoritmo basato sulla seconda tipologia di iterazione come nel listato a fianco.

In questo caso risulta fondamentale inizializzare correttamente la variabile Continua: se fosse impostata inizialmente a NO, il ciclo non avrebbe mai inizio. Nella prima modalità di ciclo introdotta la sua inizializzazione, seppur consigliabile, non era obbligatoria perché il valore della variabile Continua veniva impostato comunque dall'utente prima della verifica della condizione di uscita.

Salti incondizionati

Il salto da un punto dell'algoritmo a uno precedente può avvenire anche senza il verificarsi di una condizione precisa; si parla in questo caso di **salto incondizionato**, ovvero senza condizioni specifiche che lo determinano.
La parola chiave che rappresenta il salto incondizionato nello pseudo codice è **Goto**.

Goto: "vai a".



Nello pseudocodice si indica mediante una etichetta (numerica o testuale) il punto da cui il codice deve ripartire; all'istruzione **GOTO** segue il nome dell'etichetta corrispondente.

La sequenza di istruzioni memorizza l'etichetta (*Label*) associata a un preciso punto dell'algoritmo; quando il **GOTO** indica l'etichetta, il flusso dell'algoritmo riparte da quel punto.

Nei flow chart il salto incondizionato si rappresenta semplicemente con una freccia che rimanda a un pallino vuoto che funge, analogamente ai cicli, da etichetta identificatrice di un punto preciso del codice.

L'uso di **GOTO** nella programmazione può generare diversi problemi:
- cicli infiniti;
- limitata leggibilità del codice;
- limitata modificabilità del codice.

Il suo utilizzo è sconsigliato.
La leggibilità di un codice che racchiude un numero significativo di salti incondizionati è complessa, poco intuitiva e può produrre errori.

UTILE A SAPERSI

Un'istruzione deleteria

"*Goto statement is harmful*" ovvero "*L'istruzione Goto è deleteria per la programmazione*".

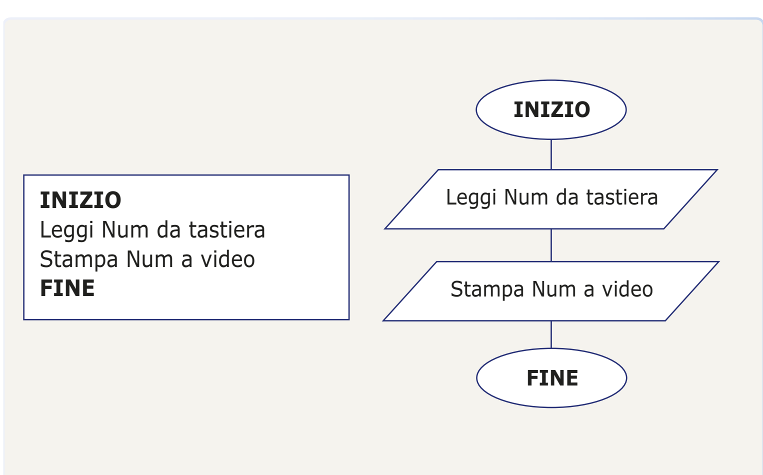
Così è commentato l'uso del GOTO nella programmazione in un celebre articolo di Dijkstra del 1968, considerato come una delle pubblicazioni alla base della programmazione strutturata.

Un teorema, enunciato dagli informatici Corrado Böhm e Giuseppe Jacopini, afferma che la programmazione può fare a meno dell'uso dei salti incondizionati; si può quindi affrontare la risoluzione di qualunque problema mediante l'uso dei costrutti precedentemente enunciati senza utilizzare i salti incondizionati; si parla, in questo caso, di *programmazione strutturata*.

Il **teorema di Böhm-Jacopini** afferma che qualunque algoritmo può essere scritto mediante il solo uso di sequenze di istruzioni elementari, selezioni e cicli condizionati.
La **programmazione strutturata** è lo stile di programmazione che si attiene alle indicazioni del teorema.

Sulla programmazione strutturata si basano tutte le tecniche di programmazione moderne; nonostante i linguaggi supportino ancora l'istruzione **GOTO**, la pratica ne sconsiglia l'utilizzo.

Sottoprogrammi



Consideriamo il semplice algoritmo, rappresentato nello schema, contenente un'operazione di lettura e una di stampa; in entrambi i casi è specificato il supporto: tastiera per l'input e video per l'output.

Tali supporti per l'input e l'output sono stati specificati come **parametri di una funzione** Stampa nel relativo blocco nel flow chart nello pseudocodice.

La *funzione* o *sottoprogramma* si definisce come segue:

La **funzione** (o **sottoprogramma**) è un blocco di istruzioni che, fornito un insieme di dati in input, svolge un compito prefissato ed eventualmente restituisce un insieme di dati in output.

Approfondimento
• Le funzioni di lettura e scrittura

In molti casi le funzioni contengono porzioni di algoritmo che risulta significativo aggregare e riutilizzare per evitare ripetizioni; vengono descritte con un flow chart (o una porzione di pseudocodice) separato da quello principale.

ESEMPIO Una sequenza di istruzioni può svolgere il compito di calcolare la media dei voti degli studenti; se nel programma è necessario ricalcolare più volte tale media, è opportuno evitare di scrivere il medesimo codice più volte; conviene piuttosto definire una funzione che svolga quel compito e richiamarla più volte, eventualmente con valori di ingresso diversi.

Subroutine: "sottoprogramma".

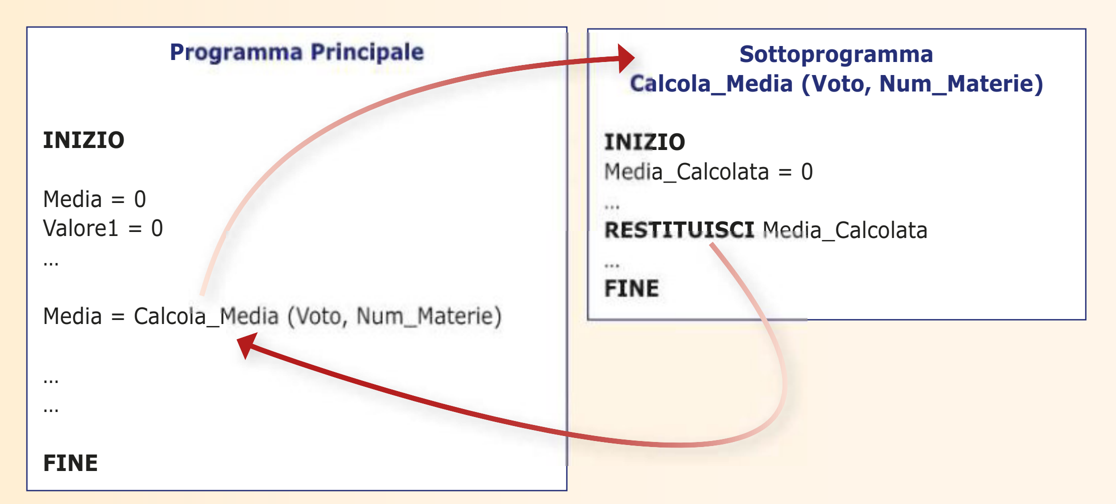
Una funzione (o subroutine) è caratterizzata da dati in ingresso e dati in uscita; può dunque svolgere il ruolo di programma a sé stante.

Per richiamare la funzione o sottoprogramma da un altro programma la notazione più usata è:

Uscita = Nome_Funzione (Ingressi)

Gli ingressi si riportano tra parentesi dopo il nome della funzione; i dati in uscita vengono assegnati alla variabile Uscita. La funzione contiene la parola chiave **RESTITUISCI** (*Return*), che indica quale valore viene restituito dalla funzione al programma chiamante. Il valore restituito è tipicamente contenuto in una variabile definita, e opportunamente inizializzata, all'interno della funzione.

ESEMPIO **Media = Calcola_Media (Voto, Num_Materie)**
La funzione **Calcola_Media** ( ) prende in ingresso il voto di una nuova materia e restituisce la media aggiornata dei voti delle varie materie (vedi schema seguente). La funzione **Calcola_Media** restituisce un valore al programma principale mediante la parola chiave **RESTITUISCI** (*Return*).



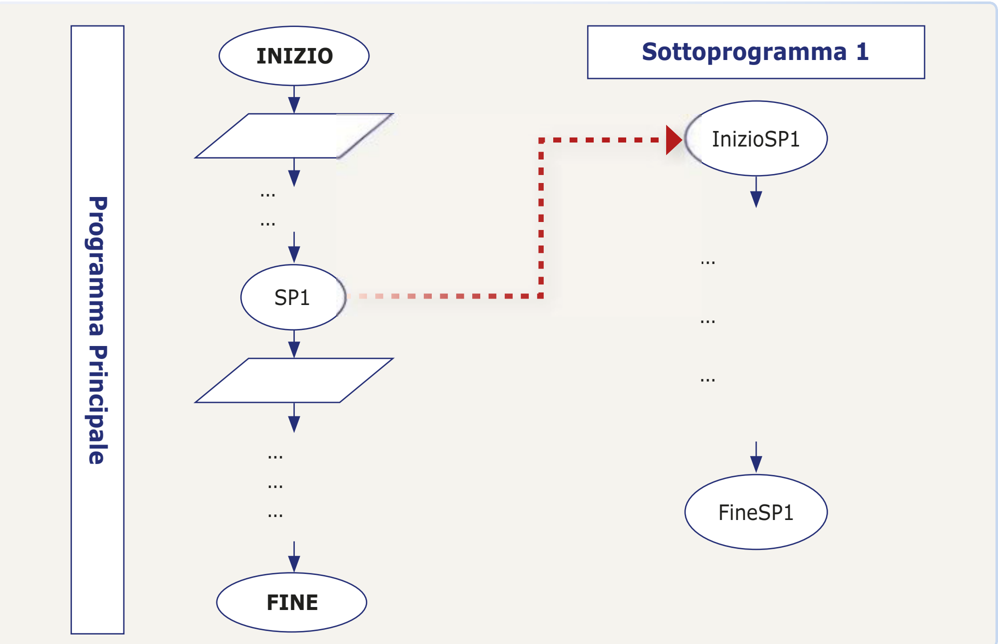
Se una funzione non restituisce alcun valore, il suo richiamo è eseguito scrivendo semplicemente il suo nome seguito da parentesi vuote "( )".

UTILE A SAPERSI

Per una migliore leggibilità

Abbiamo incontrato alcune funzioni che gestiscono l'imput e l'output: **Leggi**, **Scrivi**, **Stampa** e **Restituisci**. Per migliorare le leggibilità nei flow chart e negli pseudocodici esse sono state espresse in linguaggio naturale anziché nella notazione standard; per esempio, **Stampa Numero a video** anziché **Stampa (Numero, Video)** per indicare la stampa a video della variabile **Numero**.

Nei flow chart i sottoprogrammi si rappresentano mediante un apposito blocco che rimanda al nome del sottoprogramma da eseguire. La linea rossa a tratteggio indica come prosegue il flusso dell'algoritmo in presenza di un rinvio da un sottoprogramma.



ESERCITAZIONE GUIDATA 1

Cicli ed espressioni condizionali

Un'azienda vuole vendere, all'inizio dell'anno, un impianto acquistato alcuni anni prima e parzialmente ammortizzato. Scrivere un algoritmo tenendo presente i seguenti dati forniti in ingresso:

- *la data dell'anno in cui il bene è stato acquistato;*
- *il costo di acquisto;*
- *la percentuale annua di ammortamento;*
- *la data dell'anno in cui è bene è stato venduto;*
- *il prezzo di cessione.*

Si chiede di progettare un algoritmo che individui l'eventuale plusvalenza o minusvalenza derivante dalla cessione.

```
INIZIO
Storico = 0
Prezzo = 0
Ammortamento = 0
Risultato = "Pareggio"
Data_acquisto = Data_Corrente
Leggi Storico
Leggi Prezzo
Leggi Ammortamento
Leggi Data_Acquisto
Anni = (Data_Corrente – Data_Acquisto) / 365
Totale = Prezzo – (Storico – (Storico * (Ammortamento / 100) * Anni))
SE Totale > 0
    ALLORA
        Risultato = "Plusvalenza"
    ALTRIMENTI
        SE Totale < 0
        ALLORA
            Risultato = "Minusvalenza"
        Fine SE
Fine SE
Stampa Risultato
FINE
```

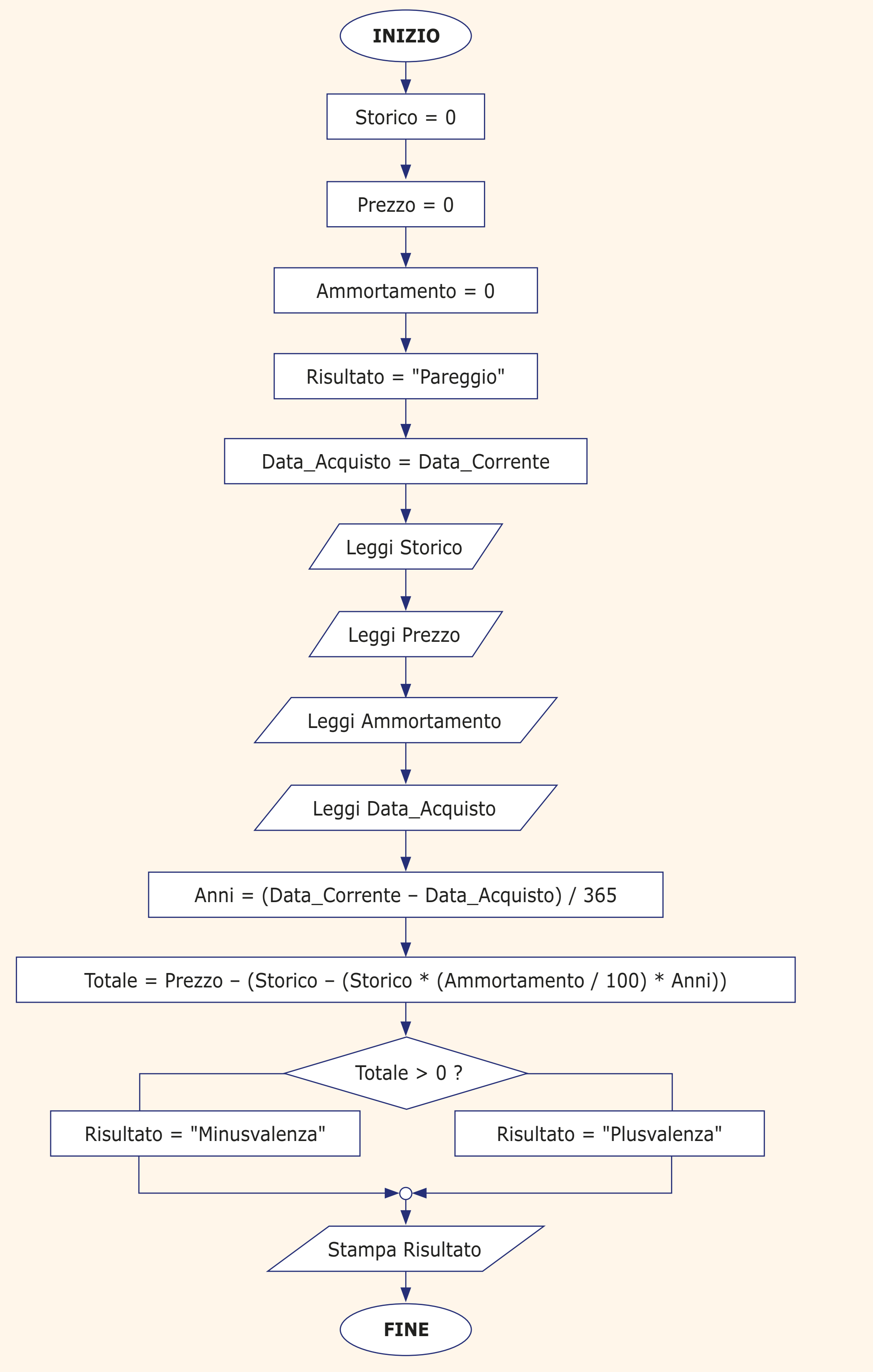
Il programma legge in ingresso la data di acquisto dell'impianto inserita dall'utente, il costo storico (costo d'acquisto originario del bene), il valore di cessione (prezzo di vendita) e la percentuale annua di ammortamento e li memorizza nelle rispettive variabili.
Prima della lettura dei dati di ingresso le variabili utilizzate sono state opportunamente inizializzate. Il numero di anni trascorsi dall'acquisto si ricava mediante l'operazione di modulo, che dà il resto della divisione intera (Data_Corrente – Data_Acquisto) / 365 (numeri di giorni in un anno).

Esempio numerico

Dati
Costo storico = euro 2000
Percentuale di ammortamento: 20%
Anni di utilizzo del bene: 3
Valore di cessione: euro 1000

Calcolo
Fondo ammortamento = euro (2000 x 20%) x 3 = euro 1200
Valore contabile: euro (2000 – 1200) = euro 800
Totale: (1000 – 800) = euro 200 → plusvalenza

ESERCITAZIONE GUIDATA 1



La percentuale di ammortamento viene applicata al valore storico (pesandola per il numero di anni trascorsi) per ricavare il fondo ammortamento e successivamente il valore residuo contabile (differenza tra costo storico di acquisto e fondo ammortamento). Infine viene calcolata la differenza tra il prezzo di cessione e il valore residuo contabile, memorizzata nella variabile Totale; se il risultato finale è positivo si indica plusvalenza; se è negativo, minusvalenza.

ESERCITAZIONE GUIDATA 2

Programmi e sottoprogrammi

Progettare un algoritmo per il calcolo del riparto delle spese sostenute per la gestione e la manutenzione di un ascensore tra le diverse famiglie di un condominio, tenendo conto sia del piano in cui abitano sia del numero di componenti il nucleo familiare.
L'algoritmo deve essere applicabile a una qualunque tipologia di condominio e fornire la soluzione mediante pseudocodice e flow chart.

Programma principale

```
INIZIO
Base = 0
Componenti = 0
Piano = 0
Base = Calcola_Base ()
Leggi Componenti
Leggi Piano
Scrivi Piano * Componenti * Base
FINE
```

Sottoprogramma Calcola_Base

```
INIZIO
Totale_Spese = 0
Denominatore = 0
Continua = "SI"
Base = 0
Leggi Totale_Spese
RIPETI
        Leggi Piano
        Leggi Componenti
        Denominatore = Denominatore + (Piano * Componenti)
        Leggi Continua // Nuovo_Nucleo_Familiare?
FINCHÉ Continua = "NO"
SE Denominatore > 0
  ALLORA
        Base = Totale_Spese / Denominatore
        Restituisci Base
  ALTRIMENTI
Fine SE
        Stampa Errore
FINE
```

La soluzione proposta prevede l'uso di una funzione che calcola la spesa base pro capite in relazione al piano e al numero dei componenti di ciascun nucleo. La funzione per il calcolo della base richiede in ingresso due variabili, opportunamente inizializzate prima dell'inizio del programma: **Totale_Spese**, indica la spesa totale da ripartire e **Denominatore**, totalizza la somma dei componenti dei nuclei familiari in relazione al piano dell'alloggio.

Un ciclo prosegue finché l'utente non specifica di non voler più inserire un nuovo nucleo. Per ogni nuovo nucleo, il piano e il numero di componenti vengono prima memorizzati e poi moltiplicati e sommati al totale corrente della somma memorizzata nella variabile Denominatore.

Quando il ciclo termina il totale della spesa viene diviso per la somma pesata al fine di ottenere la base pro capite da restituire all'algoritmo principale.

L'algoritmo principale prende in ingresso i dati relativi a uno specifico nucleo (numero di componenti e piano) e restituisce il relativo totale di spesa moltiplicando la base per il piano e per il numero di componenti.