Tipo di uscita

Docente/i accompa

Classe 1I B Ec

Località

Sito/azienda visitat

Inizio e termine atti

Mezzo di trasporto

Consenso docen
I docenti in servizi
prendono atto e a
Ruggeri Luca
De Luca
Cavaliere
Caretto

Costi
Costo a carico alli

Unità 5 Primi elementi di programmazione

1 Introduzione alla programmazione

Quante volte ti è capitato di utilizzare il computer di casa per fare "girare" i tuoi programmi, per esempio un videogioco appena regalato oppure un software di videoscrittura per impaginare la relazione di storia o, ancora, un browser per navigare su Internet e visitare le pagine Web che più ti piacciono? Quasi ogni giorno, probabilmente.

Ti sei mai chiesto che cosa succede dentro il computer quando "lanci" un programma facendo doppio clic sulla sua icona? O quando carichi un gioco sulla console mentre ti prepari a impugnare il controller remoto?

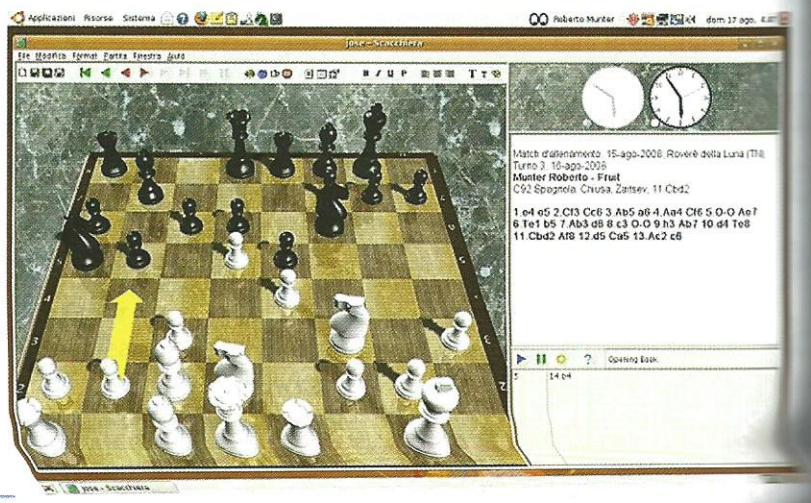
Devi sapere che ogni programma è costituito da tante, tantissime istruzioni scritte una dopo l'altra, una lista delle cose da fare che il microprocessore del dispositivo utilizzato legge ed esegue in sequenza ad altissima velocità, in modo da elaborare i dati di ingresso per fornire i risultati dell'elaborazione richiesti.

Esempio 1

Prendiamo come esempio il programma che ci permette di giocare a scacchi contro il computer.

Una volta lanciata l'esecuzione del programma, le prime istruzioni che il microprocessore deve eseguire sono quelle che hanno lo scopo di disegnare a video la scacchiera con i pezzi posizionati a inizio partita. Seguiranno poi in sequenza nel programma le istruzioni necessarie a richiedere al giocatore il suo nome e se vuole giocare con i pezzi bianchi o neri. Una volta forniti in ingresso questi dati iniziali – attraverso la tastiera e un clic del mouse – saranno richieste in ingresso al giocatore le coordinate della sua prima mossa. Anche in questo caso, attraverso il mouse, il giocatore cliccherà sul pezzo che desidera muovere e sulla casella su cui vuole eseguire lo spostamento. Il programma risponderà eseguendo le istruzioni che disegnano a video la scacchiera con il pezzo selezionato nella nuova posizione e, in successione, le istruzioni per decidere la sua prossima mossa.

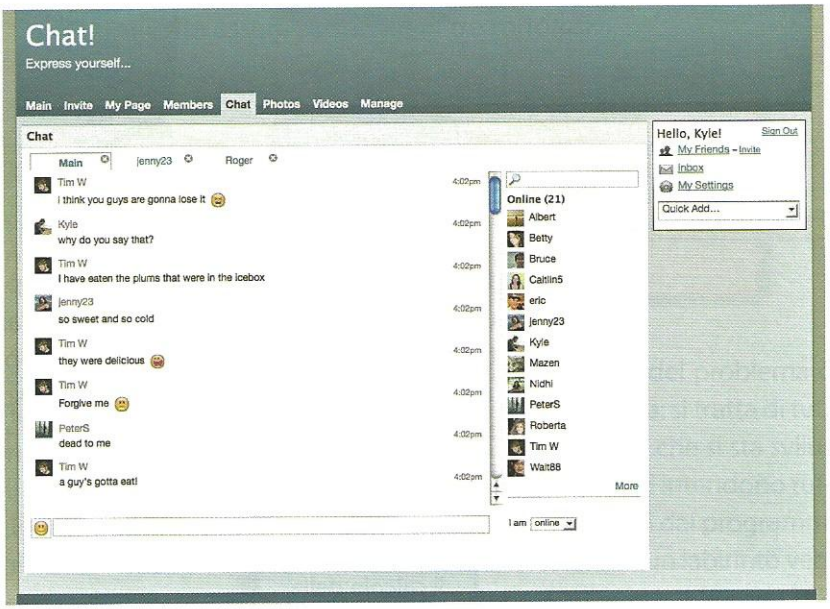
E così via, mossa dopo mossa sino a fine partita o sino a quando il giocatore avrà voglia.



Videata di un programma per giocare a scacchi al computer.

Esempio 2

Un altro esempio ci può essere fornito dal programma che ci permette di "chattare" in Internet. Una volta lanciata l'esecuzione dell'applicazione, le prime istruzioni che vengono eseguite richiedono all'utente il suo nickname e la sua password, che sono inseriti attraverso la tastiera. Dopo che le istruzioni del programma di "chat" avranno verificato la correttezza dei dati inseriti, l'utente avrà accesso al sito e potrà sapere quali sono i suoi amici on line e, attraverso il mouse e la tastiera, interagire con loro. Saranno le istruzioni del programma che, una volta selezionato il contatto al quale si vuole mandare un messaggio e riempita la casella di testo, si occuperanno di farlo pervenire all'amico attraverso la rete. E così via, messaggio dopo messaggio.

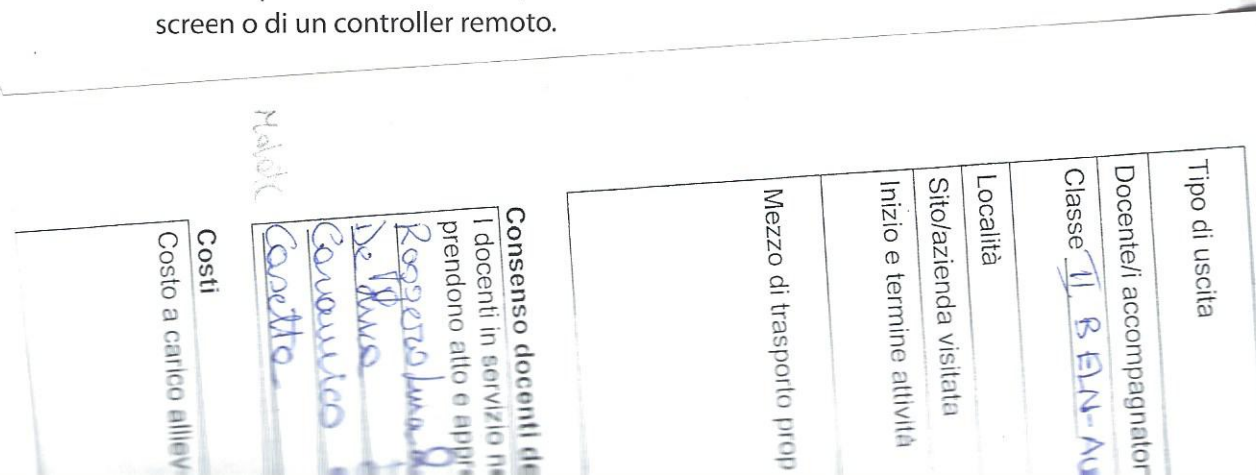


Videata di un programma di chat on line.

Possiamo quindi definire un programma come segue:

Programma **Un *programma* è un insieme finito di istruzioni che, eseguite in sequenza, permettono di elaborare i dati in ingresso per ottenere in uscita i risultati richiesti dall'elaborazione, in modo da risolvere un determinato problema.**

Nell'**esempio 1**, il problema è la partita a scacchi fra utente e computer, i dati in ingresso sono le mosse che il giocatore immette volta per volta, i risultati sono le contromosse, sino a fine partita. Nell'**esempio 2**, il problema da gestire è lo scambio di messaggi fra utenti collegati in chat, i dati in ingresso sono i nickname del mittente e del destinatario e il testo dei messaggi da inviare, i risultati sono i messaggi ricevuti dagli altri utenti connessi inviati in risposta.
In questo capitolo vogliamo iniziare a capire come si "sviluppa" un programma. Naturalmente non partiremo dai programmi che abbiamo utilizzato per gli esempi precedenti, ma da programmi molto più semplici. Inoltre, per semplificare ulteriormente lo studio, chiameremo in generale "elaboratore" il computer a disposizione – portatile, desktop, netbook, smartphone, tablet o console di gioco che sia – con il quale comunichiamo per mezzo di una tastiera, di un mouse, di un touch screen o di un controller remoto.



L'elaboratore è quindi l'esecutore delle istruzioni dei nostri programmi, istruzioni che elaborano i dati in ingresso per fornire, al termine dell'elaborazione, i risultati in uscita (fig. 1).

Figura 1

Supponiamo, per esempio, di voler sviluppare un programma per il calcolo della media aritmetica di 3 numeri, cioè un programma che richiede all'utente in ingresso da tastiera 3 numeri e, al termine degli input, ne calcola la media aritmetica e la visualizza a video. Se fornissimo in ingresso i numeri 10, 6 e 5, la loro media aritmetica sarebbe data dalla somma $10 + 6 + 5 = 21$ che divisa per 3 ci fornirebbe il risultato 7 (fig. 2).

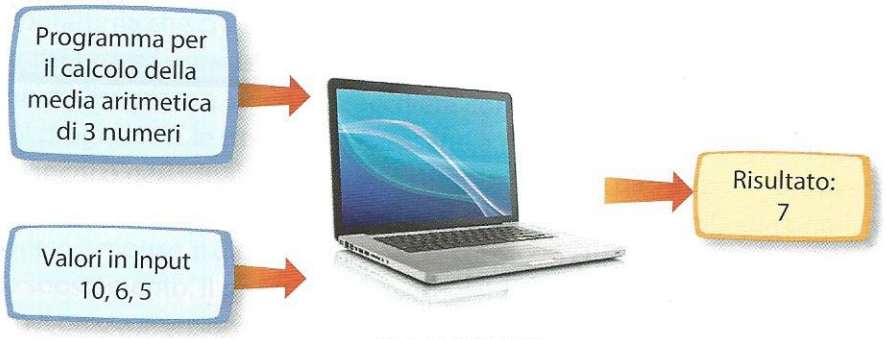


Figura 2

Quello che ora vogliamo fare è passare *dalla descrizione a parole del problema* "calcolare la media aritmetica di 3 numeri" *all'insieme di istruzioni che costituiscono il programma*: in questo modo, fornito all'elaboratore il programma e i 3 numeri in input da tastiera, l'elaboratore può calcolare e fornire in uscita la media richiesta.

2 Dal problema al programma

Per passare dal problema da risolvere al programma che lo risolve "automaticamente" occorre quindi **formalizzare il problema** che, espresso inizialmente in linguaggio naturale, deve passare attraverso varie *fasi di trasformazione* che permettano di focalizzare meglio gli obiettivi e il modo per raggiungerli, fino a ottenere il programma che risolve il problema di partenza.

Figura 3
Fasi del processo per passare dal problema al programma.

In **figura 3** sono riassunte alcune tra le principali fasi che costituiscono il processo di formalizzazione e che applicheremo negli esempi successivi.
Il primo passo da affrontare consiste nell'**analisi** approfondita del problema che si vuole risolvere. In questa fase occorre porre molta attenzione a ciò che ci viene richiesto, per essere sicuri di aver capito esattamente *qual è l'obiettivo che ci poniamo e che cosa abbiamo a disposizione per raggiungerlo*. Più dettagliatamente, la fase di analisi può a sua volta essere suddivisa in tre *sottofasi*:

Il primo passo dell'analisi consiste, dopo una lettura approfondita del problema da risolvere, nell'individuazione dei **dati in ingresso** (**input**) al problema: si tratta di tutte le informazioni che devono essere fornite in ingresso al programma che si sta sviluppando, con gli eventuali vincoli di integrità. Per **vincoli di integrità** si intendono tutte quelle condizioni che gli input devono rispettare per essere accettati dal programma. Se, per esempio, in un videogioco possiamo scegliere di spostarci in un labirinto verso destra o verso sinistra, con la tastiera non potremo che premere i tasti freccia ← o → perché saranno gli unici tasti considerati validi (**fig. 4**).
Un ulteriore esempio potrebbe essere il seguente: supponiamo di dover fornire in ingresso la nostra data di nascita nel formato giorno/mese/anno per iscriverci a un sito. I vincoli che questi input devono rispettare per essere accettati dal programma impongono che il valore che indica il mese sia compreso tra 1 e 12, il valore che indica il giorno sia compreso tra 1 e il numero di giorni del mese corrispondente, mentre per il valore che indica l'anno il vincolo dipenderà dal problema che stiamo affrontando.
Il secondo passo dell'analisi consiste nell'individuazione dei **dati in uscita** (**output**), cioè dei risultati attesi dall'elaborazione dei dati in ingresso da parte del programma. Anche per gli output devono essere individuati gli eventuali vincoli ai quali essi devono sottostare per essere considerati validi.

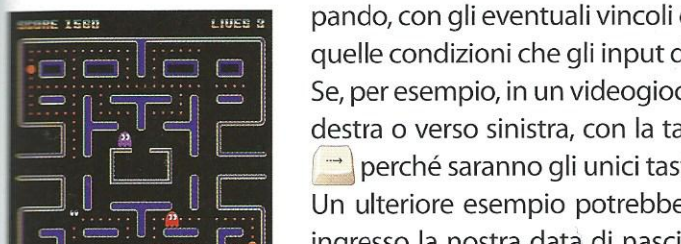


Figura 4
Lo screenshot del famosissimo labirinto di PacMan, dove ci si muove con i tasti freccia.

Se, per esempio, abbiamo raggiunto il punteggio necessario per passare al livello successivo di un videogioco e invece di cambiare schema ritorniamo a quello precedente, significa che il programma non ha rispettato il vincolo posto al risultato che abbiamo ottenuto (**fig. 5**).

Figura 5
Candy Crush offre ai suoi giocatori centinaia di livelli di gioco.

Tipo di uscita	
Docente/i accom	
Classe	
Località	
Sito/azienda vis	
Inizio e termine	
Mezzo di traspo	

Consenso dei
I docenti in se prendono atto

Costi
Costo a carico

O ancora, se abbiamo un programma che deve ordinare alfabeticamente (dalla A alla Z) un elenco di cognomi e in uscita otteniamo invece come risultato l'elenco ordinato in modo decrescente (dalla Z alla A), allora significa che l'ordinamento effettuato dal programma non ha rispettato il vincolo posto all'output.

L'ultima sottofase dell'analisi consiste nell'individuazione delle **relazioni tra i dati in input e quelli in output** (**I/O**), cioè dei *collegamenti logici esistenti tra le informazioni in ingresso e quelle in uscita dal programma*.

La relazione tra I/O può essere una formula matematica che lega i dati in ingresso al risultato atteso, come nel caso del calcolo della media aritmetica, oppure un modello logico che permette di approfondire il legame esistente tra I/O per facilitare poi le fasi successive del processo di formalizzazione.

Per capire meglio ciò che si intende per relazione tra i dati in input e quelli in output, applichiamo la metodologia vista ad alcuni semplici problemi, di cui sviluppiamo l'analisi.

Problema 1

Calcolare la media aritmetica di 3 numeri in ingresso.

ANALISI

Dati in input	I 3 valori numerici di cui si vuole calcolare la media aritmetica.
Dati in output	La media aritmetica.
Relazione tra I/O	La media aritmetica è data dalla somma dei 3 valori in ingresso divisa per 3:

$$Media = \frac{1°valore+2°valore+3°valore}{3}$$

Problema 2

Forniti in ingresso al programma la base e l'altezza di un triangolo, calcolarne l'area.

ANALISI

Dati in input	La lunghezza della base e la lunghezza dell'altezza del triangolo; il vincolo da porre è accettare in ingresso solo valori maggiori di 0.
Dati in output	L'area del triangolo.
Relazione tra I/O	L'area del triangolo si ottiene moltiplicando la base per l'altezza e dividento per 2:

$$Area = \frac{Base*Altezza}{2}$$

dove il simbolo * indica, nel campo informatico, la moltiplicazione.

Problema 3

Fornito in ingresso al programma il numero di lati (si suppone compreso tra 3 e 5) di un poligono regolare e la lunghezza del lato del poligono, indicare di quale poligono si tratta e calcolarne il perimetro.

ANALISI

Dati in input	Il numero di lati del poligono regolare e la lunghezza del lato del poligono. I vincoli da porre sono che il numero di lati sia un valore intero positivo compreso tra 3 e 5 e la lunghezza del lato sia un valore maggiore di 0.
Dati in output	La stampa a video del tipo di poligono regolare e del suo perimetro.
Relazione tra I/O	– Se il numero di lati è uguale a 3 allora è un "Triangolo equilatero"; – se il numero di lati è uguale a 4 allora è un "Quadrato"; – se il numero di lati è uguale a 5 allora è un "Pentagono". Il perimetro si calcola moltiplicando il numero di lati per la lunghezza del lato: *Perimetro = NumeroLati * LunghezzaLato*

Problema 4

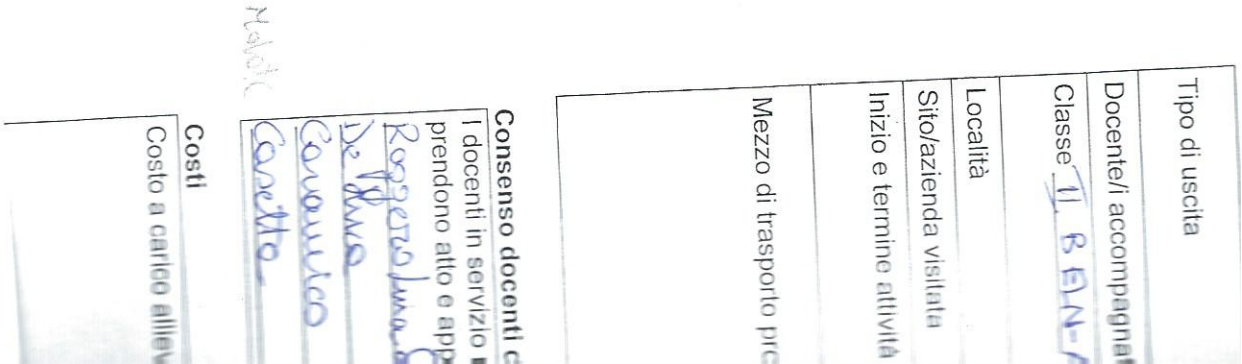
Determinare il valore più grande tra 3 valori numerici in ingresso.

ANALISI

Dati in input	I 3 valori numerici: si può supporre di non avere vincoli, quindi di accettare in ingresso sia numeri interi sia numeri con la virgola, positivi e negativi.
Dati in output	Il valore più grande fra i 3.
Relazione tra I/O	– Se il 1° valore è maggiore del (o uguale al) 2° valore e se il 1° valore è maggiore del (o uguale al) 3° valore allora il valore maggiore è il 1°; – se il 2° valore è maggiore del (o uguale al) 1° valore e se il 2° valore è maggiore del (o uguale al) 3° valore allora il valore maggiore è il 2°; – se il 3° valore è maggiore del (o uguale al) 1° valore e se il 3° valore è maggiore del (o uguale al) 2° valore allora il valore maggiore è il 3°.

Nei problemi analizzati, dopo aver individuato i dati in ingresso e quelli in uscita al problema si è cercata la relazione che li lega; nei primi due problemi il legame tra I/O è fornito dalla formula matematica, mentre negli ultimi due problemi la relazione tra I/O fa riferimento al confronto logico tra i dati in ingresso per arrivare a individuare i dati in uscita.

Al termine della fase di analisi, il problema da risolvere dovrebbe risultare più chiaro e dettagliato, pronto quindi per la più delicata e complessa fase successiva, in cui si giunge a rappresentare il problema di partenza attraverso l'**algoritmo risolutore** corrispondente.



Tipo di uscita

Docente/i accompagnat

Classe 1I B EN-A

Località

Sito/azienda visitata

Inizio e termine attività

Mezzo di trasporto pro

Consenso docenti d

I docenti in servizio r
prendono atto e app

Costi

Costo a carico allie

3 Lo sviluppo dell'algoritmo

In questa fase, partendo dai risultati ricavati nell'analisi svolta precedentemente e, in particolar modo, dalle relazioni tra I/O, si tratta di individuare un *metodo ottimale di risoluzione del problema di partenza*, cioè una sequenza di azioni che, seguendo le indicazioni date dalla relazione tra I/O, permetta di trasformare i dati in ingresso in risultati, cioè in dati di uscita.

L'elaboratore non è in grado autonomamente di trovare il metodo risolutore: il suo compito infatti consiste nell'eseguire in sequenza le istruzioni del programma da elaborare una dopo l'altra, sino al loro termine. L'elaboratore non pensa, non sa ragionare. Non conosce le regole per il calcolo della media aritmetica, non conosce le regole del gioco degli scacchi, non sa controllare se possiamo cambiare livello di gioco o se abbiamo perso.

Sta al programmatore il compito di "indicare" all'elaboratore il metodo risolutore del problema, le azioni da eseguire e le regole da seguire per ottenere il risultato voluto, cioè l'algoritmo risolutore del problema di partenza. Quindi il programmatore è colui che "trasmette" all'elaboratore il proprio ragionamento sotto forma di algoritmo, che tradotto in un programma potrà essere eseguito.

Esempio 3

I dati in imput: gli ingredienti.

Il risultato: il tiramisù.

Un semplice esempio di algoritmo, slegato dall'ambito informatico e preso in prestito dal mondo culinario, è fornito dal concetto di ricetta. Infatti una ricetta non è altro che una sequenza finita di azioni (sbatti le uova, aggiungi lo zucchero, inzuppa i savoiardi ecc.) che permettono di risolvere un problema (preparare il "tiramisù") trasformando i dati in ingresso (uova, zucchero, caffè ecc.) in risultati (il dolce pronto per essere mangiato). Ecco la sequenza delle azioni:

1. Preparare il caffè.
2. Separare i tuorli dagli albumi e montare i tuorli insieme allo zucchero.
3. Unire poi il mascarpone amalgamandolo bene.
4. Montare gli albumi a neve ben ferma e aggiungerli con delicatezza alla crema di uova e mascarpone.
5. Immergere velocemente i savoiardi nel caffè e coprire il fondo di una vaschetta con uno strato di savoiardi.
6. Stendere uno strato di crema al mascarpone e procedere con un secondo strato di biscotti e coprire con la rimanente crema.
7. Al termine, coprire con la crema rimanente e spolverare di cacao.

Esempio 4

$$\begin{array}{cccc} & & {}^{1} & \\ 1 & 2 & 8 & + \\ 3 & 5 & 7 & = \\ \hline \dots & 8 & 5 & \end{array}$$

L'addizione di due numeri viene eseguita applicando un algoritmo in cui i due addendi sono i dati in ingresso e la somma è il risultato finale. Iniziando da destra, si sommano le 2 cifre corrispondenti alle unità. Se la somma ottenuta è maggiore di 9 allora si sottrae 10 tenendo il resto ottenuto e andando poi a ricordare il riporto nella prossima somma. Si prosegue spostandosi verso sinistra sommando le cifre corrispondenti alle decine – con l'eventuale riporto – e si applica lo stesso criterio già visto ciclicamente sino a fine addizione.

Una semplice addizione richiede l'esecuzione di un complesso algoritmo che ripete ciclicamente la somma di coppie di cifre da destra verso sinistra.

Figura 6
Il percorso più rapido per andare da Milano a Roma.

Il concetto di algoritmo è applicato in molte discipline, ma è nell'informatica, e in particolare nella programmazione, che assume una notevole importanza, perché è in questa delicata fase del processo di formalizzazione che sono richieste capacità di astrazione tali da individuare non solo l'algoritmo risolutore del problema, ma soprattutto l'algoritmo risolutore *ottimale*, che segue criteri di generalità e di sintesi. Per chiarire questo punto, vediamo alcuni semplici esempi.
Se devo andare in auto da Milano a Roma, posso scegliere tra varie combinazioni di strade: se non ho fretta posso passare da Venezia scendendo lungo l'Adriatico oppure da Torino passando lungo la costa ligure e toscana per poi raggiungere Roma. Tuttavia, la strada migliore resta quella che da Milano passa da Bologna, perché è la più diretta, veloce e meno costosa (**fig. 6**).

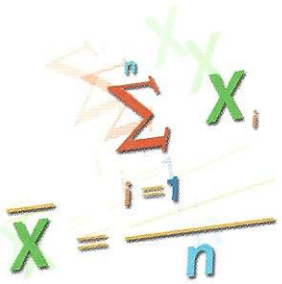
Veniamo ora a esempi più legati all'ambito informatico.
Se devo ordinare alfabeticamente un elenco di cognomi, posso cercare dapprima tutti quelli che iniziano con la lettera A e poi tutti quelli che iniziano con la lettera B e così via sino alla lettera Z. Ma non è forse più conveniente spostare verso le prime posizioni dell'elenco i cognomi che iniziano con le lettere comprese tra la A e la F e, contemporaneamente, spostare verso le ultime posizioni quelli che iniziano con le lettere tra la P e la Z, lasciando a metà elenco gli altri cognomi? Con una seconda selezione è probabilmente possibile ordinare del tutto l'elenco, mentre con l'altro metodo sarei probabilmente a metà alfabeto.
Un altro caso interessante è il seguente: se devo cercare il mio cognome in un elenco di persone invitate a una festa a cui partecipano centinaia di ragazzi e ragazze, mi conviene scorrere l'elenco degli invitati ordinato alfabeticamente, oppure l'ordinamento in una ricerca non ha importanza? Provate a pensarci, la risposta non è poi così difficile!

Questi sono solo alcuni esempi che mettono in evidenza la complessità che contraddistingue la fase di sviluppo dell'algoritmo nel processo di formalizzazione.

Possiamo a questo punto definire un algoritmo nel seguente modo:

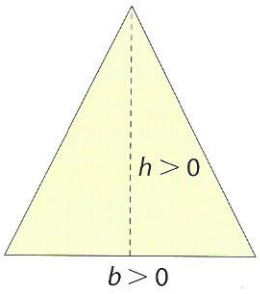
Algoritmo **Un *algoritmo* è un insieme finito di azioni che risolvono un determinato problema, trasformando i dati di input in dati di output (o risultati) attraverso le relazioni esistenti tra input e output.**

Vediamo ora come esempio lo sviluppo di un algoritmo che risolve il problema:
Calcolare la media aritmetica di 3 numeri in ingresso
di cui abbiamo già visto la fase dell'analisi:



1. Inizio
2. Ricevi in ingresso da tastiera i tre valori
3. Calcola la media sommando i tre valori in ingresso e dividendo la somma ottenuta per 3
4. Stampa a video la media aritmetica calcolata
5. Fine

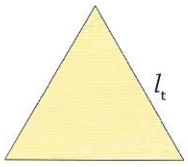
La sequenza di passi da compiere per ricavare un algoritmo che risolva il problema è chiara e le azioni, eseguite una dopo l'altra, ci portano a calcolare la media aritmetica in modo molto semplice e diretto. Se infatti fornissimo in ingresso da tastiera i tre valori 15, 5, 40, si otterrebbe come risultato:
20 (infatti, 15 + 5 + 40 = 60 e 60 : 3 = 20)
che rappresenta la media aritmetica cercata.
Passiamo ora a sviluppare l'algoritmo risolutore del problema:
Forniti in ingresso al programma la base e l'altezza di un triangolo, calcolarne l'area
di cui abbiamo già visto la fase dell'analisi:

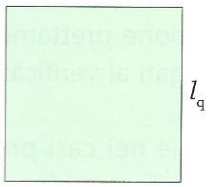


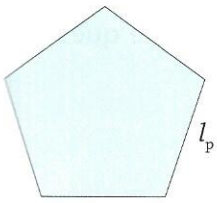
1. Inizio
2. Ricevi in ingresso da tastiera la lunghezza della base del triangolo (con il vincolo che sia maggiore di 0)
3. Ricevi in ingresso da tastiera la lunghezza dell'altezza del triangolo (con il vincolo che sia maggiore di 0)
4. Calcola l'area del triangolo moltiplicando la lunghezza della base per la lunghezza dell'altezza e dividendo il prodotto ottenuto per 2
5. Stampa a video l'area del triangolo calcolata
6. Fine

Anche in questo caso la sequenza di passi da compiere per ricavare un algoritmo che risolva il problema è chiara e le azioni, eseguite una dopo l'altra, ci portano a calcolare l'area del triangolo in modo molto semplice e diretto. Se infatti fornissimo in ingresso da tastiera come base del triangolo il valore 12,5 e come altezza 14,2, si otterrebbe come area del triangolo il valore:
88,75 (infatti, 12,5 · 14,2 : 2 = 88,75).

Passiamo ora a sviluppare l'algoritmo risolutore del problema:
Fornito in ingresso al programma il numero di lati (si suppone compreso tra 3 e 5) di un poligono regolare e la lunghezza del lato del poligono, indicare di quale tipo di poligono si tratta e calcolarne il perimetro.







1. Inizio
2. Ricevi in ingresso da tastiera il numero di lati del poligono (con vincolo che sia intero e compreso tra 3 e 5)
3. Ricevi in ingresso da tastiera la lunghezza del lato (con vincolo che sia maggiore di 0)
4. Se il numero di lati del poligono è uguale a 3 allora prosegui, altrimenti salta all'azione 7
5. Stampa a video la parola "Triangolo equilatero"
6. Salta all'azione 11
7. Se il numero di lati del poligono è uguale a 4 allora prosegui, altrimenti salta all'azione 10
8. Stampa a video la parola "Quadrato"
9. Salta all'azione 11
10. Stampa a video la parola "Pentagono"
11. Calcola il perimetro del poligono regolare moltiplicando il numero di lati del poligono per la lunghezza del lato
12. Stampa a video il perimetro del poligono regolare
13. Fine

In questo algoritmo si evidenzia come l'*esecuzione prettamente sequenziale delle azioni può essere interrotta con dei salti, legati al verificarsi o meno di particolari condizioni*: effettuando un controllo che fornisce un risultato positivo (vero) o negativo (falso) è possibile stabilire quali azioni eseguire successivamente scegliendo tra due possibili alternative.
Vediamone il funzionamento: se in input fornissimo il valore 4 (numero lati del poligono regolare) e successivamente il valore 15 (lunghezza del lato) si visualizzerebbe a video, correttamente, la parola "Quadrato" e si calcolerebbe il valore 60 (infatti, $15 \cdot 4 = 60$) corrispondente al perimetro del quadrato considerato.

Vediamo, infine, un quarto esempio di algoritmo. Il problema a cui facciamo riferimento, e di cui abbiamo già ricavato l'analisi, è il seguente:
Determinare il valore più grande tra 3 valori numerici in ingresso.

1. Inizio
2. Ricevi in ingresso da tastiera il 1° valore, il 2° valore, il 3° valore
3. Se il 1° valore è $\geq$ del 2° valore allora prosegui, altrimenti salta all'azione 9
4. Se il 1° valore è $\geq$ del 3° valore allora prosegui, altrimenti salta all'azione 7

5. Stampa a video il 1° valore in quanto è il maggiore
6. Salta all'azione 13
7. Stampa a video il 3° valore in quanto è il maggiore
8. Salta all'azione 13
9. Se il 2° valore è ⩾ del 3° valore allora prosegui, altrimenti salta all'azione 12
10. Stampa a video il 2° valore in quanto è il maggiore
11. Salta all'azione 13
12. Stampa a video il 3° valore in quanto è il maggiore
13. Fine

Anche questo esempio di algoritmo evidenzia come l'esecuzione prettamente sequenziale delle azioni può essere interrotta con dei salti legati al verificarsi o meno di particolari condizioni.
Il principale problema che emerge, come si evidenzia anche nei casi precedentemente illustrati, è la difficoltà nell'esprimere in questa forma algoritmi particolarmente complessi o con molte istruzioni: oltre a essere prolissi, si potrebbero generare delle ambiguità di interpretazione nel caso di algoritmi di maggior complessità. Vedremo più avanti come risolvere anche questo problema, quando introdurremo gli schemi di flusso.

Qualcosa in più — 9 algoritmi che hanno cambiato il futuro

Ogni giorno, attraverso l'utilizzo degli strumenti tecnologici a nostra disposizione, compiamo azioni che a noi sembrano del tutto normali senza renderci conto della complessità che sta dietro a esse. Cerchiamo informazioni in rete per mezzo di motori di ricerca, archiviamo dati su potenti hard disk o su server remoti, facciamo acquisti on line, comprimiamo file di grandi dimensioni, chattiamo con gli amici attraverso i social network. Tutto ci sembra a portata di computer... ma è solo grazie a idee fondamentali, trasformate in algoritmi risolutori, che è possibile eseguire dei compiti impensabili sino a qualche anno fa.
Di questi algoritmi che hanno cambiato i nostri comportamenti quotidiani ci parla John MacCormick, docente universitario di Informatica al Dickinson College in Pennsylvania, nel suo saggio dal titolo "*9 algoritmi che hanno cambiato il futuro*". *Google* non potrebbe essere così veloce nelle ricerche se non esistesse l'algoritmo del *PageRank*, un lettore musicale di file MP3 non potrebbe contenere centinaia di canzoni se non ci fossero gli algoritmi di compressione, l'*e-commerce* non esisterebbe se non esistesse la crittografia e gli algoritmi che la implementano... Insomma, senza algoritmi il nostro mondo, e il nostro futuro, non sarebbero quello che sono e quello che saranno.

4 Il concetto di variabile

Prima di continuare nell'analisi delle altre fasi che costituiscono il processo di formalizzazione, è opportuno approfondire un concetto basilare nella programmazione, quello di variabile.

Variabili Le ***variabili*** **sono gli oggetti utilizzati dalle istruzioni del programma. Esse risiedono nella memoria dell'elaboratore e corrispondono a contenitori dei valori che sono elaborati durante l'esecuzione del programma. Alle variabili è associato un nome, detto *identificatore*, che le individua univocamente all'interno del programma. Sono chiamate variabili perché a loro viene associato un valore che può cambiare durante l'esecuzione del programma.**

Per capire meglio, facciamo riferimento agli esempi precedenti. Nel problema del calcolo della media aritmetica di tre numeri in input, abbiamo chiamato questi input "1° valore", "2° valore" e "3° valore", mentre nell'algoritmo per il calcolo dell'area del triangolo, abbiamo genericamente parlato di "lunghezza della base" del triangolo e di "lunghezza dell'altezza" del triangolo. Per gli input nell'algoritmo sui poligoni regolari abbiamo parlato di "numero di lati" del poligono e di "lunghezza del lato" del poligono. Infine, nell'algoritmo riguardante la ricerca del valore più grande tra tre numeri ricevuti in ingresso, abbiamo chiamato gli oggetti contenenti questi input "1° valore", "2° valore" e "3° valore".
In sostanza abbiamo cercato di individuare i dati che vogliamo manipolare durante lo sviluppo dell'algoritmo associando loro dei nomi.
Osserviamo, però, che il nostro sforzo di chiarezza può essere decisamente migliorato se, al posto di usare nomi generici, identifichiamo i dati da elaborare con un nome che indichi con più precisione a cosa corrisponde il valore. Per esempio, *BaseTriangolo* sarà l'oggetto contenente il valore numerico corrispondente alla lunghezza della base del triangolo, *AltezzaTriangolo* sarà l'oggetto contenente il valore numerico corrispondente alla lunghezza dell'altezza del triangolo, *AreaTriangolo* l'oggetto contenente il valore dell'area (**fig. 7**).
Questo nome viene detto **identificatore** e ha lo scopo di *individuare in modo univoco una variabile, cioè un contenitore nella memoria dell'elaboratore e il valore in esso contenuto*:

Figura 7
Usiamo l'immagine di un bicchiere vuoto come metafora per rappresentare il concetto di variabile visto come contenitore di un valore.

Si potrebbero identificare con i nomi *A*, *B*, *C* le tre variabili contenenti i tre valori numerici in ingresso nell'algoritmo del calcolo della media aritmetica e in quello della ricerca del maggiore (**fig. 8**): a esse corrisponderanno tre distinti contenitori memorizzati nella memoria dell'elaboratore e al loro interno saranno contenuti rispettivamente i tre valori numerici introdotti da tastiera.

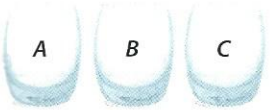


Figura 8
La rappresentazione delle variabili *A*, B e *C*.

Figura 9
La rappresentazione delle variabili dell'algoritmo su poligoni regolari.

Nel caso dell'algoritmo dei poligoni regolari, chiameremo *NumeroLati* l'identificatore della variabile contenente il numero di lati del poligono fornito in ingresso al programma, mentre l'identificatore *LunghezzaLato* corrisponderà a un'altra variabile che conterrà la lunghezza del lato del poligono regolare e, infine, il nome *Perimetro* identificherà la variabile che al suo interno conterrà il perimetro del poligono (**fig. 9**). Facendo riferimento all'identificatore della variabile, in realtà si farà riferimento al valore in essa contenuto per poterlo elaborare.

L'operazione tipica che viene eseguita su di una variabile è detta **assegnazione** e corrisponde all'inserimento di un valore nel contenitore associato all'identificatore della variabile, sostituendo il valore precedentemente contenuto. Il simbolo utilizzato negli algoritmi per indicare un'operazione di assegnazione è una freccia orientata da destra verso sinistra ← per indicare che tutto ciò che si trova a destra della freccia va assegnato al contenitore della variabile il cui identificatore si trova a sinistra della freccia, sostituendo quindi il valore precedentemente contenuto.

Se consideriamo l'algoritmo dei poligoni equilateri, l'istruzione *Calcola il perimetro del poligono equilatero moltiplicando il numero di lati del poligono per la lunghezza del lato* può essere scritta in modo molto più chiaro e sintetico (**fig. 10**):

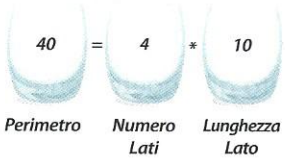


Figura 10

Perimetro ← *NumeroLati* LunghezzaLato*

che si legge:

Assegna alla variabile Perimetro il valore dell'espressione NumeroLati moltiplicato per LunghezzaLato

Se consideriamo l'algoritmo per il calcolo dell'area di un triangolo, l'istruzione:

Calcola l'area del triangolo moltiplicando la lunghezza della base per la lunghezza dell'altezza e dividendo il prodotto ottenuto per 2

può essere scritta in modo molto più chiaro (**fig. 11**):

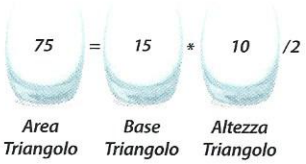


Figura 11

$$AreaTriangolo \leftarrow \frac{BaseTriangolo^{*}\ AltezzaTriangolo}{2}$$

che si legge:

Assegna alla variabile AreaTriangolo il valore dell'espressione BaseTriangolo moltiplicato per AltezzaTriangolo, il tutto diviso 2

Nella **tabella 1** sono riportate e commentate alcune semplici operazioni di assegnazione, che hanno come oggetto due variabili (corrispondenti quindi a due contenitori per valori numerici) individuate rispettivamente dagli identificatori *A* e *B*.

Assegnazione	Risultato	Commento
$A \leftarrow 9$	**Prima:** A (vuoto) **Dopo:** A 9	Nel contenitore associato all'identificatore della variabile A viene inserito il valore 9. Nel caso in cui il contenitore associato ad A contenga un precedente valore, questo viene perso e sostituito dal nuovo valore.
$B \leftarrow A$	**Prima:** A 9, B (vuoto) **Dopo:** A 9, B 9	Nel contenitore identificato da *B* e corrispondente alla variabile B viene inserito il valore contenuto nel contenitore associato alla variabile *A*. In questo caso anche *B* assume il valore 9. Il contenitore associato ad *A*, in questo caso, non perde il suo valore di partenza.
$A \leftarrow A+1$	**Prima:** A 9 **Dopo:** A 10	È calcolato il valore dell'espressione a destra della freccia, leggendo il contenuto del contenitore identificato da *A* e aumentandolo di una unità, per poi inserire il risultato ottenuto nuovamente nel contenitore della variabile *A*. La variabile *A* perde il vecchio valore che è sostituito con quello nuovo.
$A \leftarrow A+B$	**Prima:** A 10, B 9 **Dopo:** A 19, B 9	Il contenuto della variabile *A* è addizionato al contenuto della variabile *B* e il risultato dell'espressione è inserito nel contenitore della variabile *A* (a sinistra della freccia). La variabile *A* perde il vecchio valore che viene sostituito con quello nuovo, mentre la variabile *B* mantiene il suo valore inalterato.
$AUS \leftarrow A$ $A \leftarrow B$ $B \leftarrow AUS$	**Prima:** A 19, B 9, AUS (vuoto) **Dopo:** A 9, B 19, AUS 19	Nello scambio di *A* con *B* occorre utilizzare una variabile ausiliaria, per poter scambiare il valore contenuto nelle variabili *A* e *B*. In questo modo non si perde nessun valore.

Tabella 1
Alcune operazioni di assegnazione.

5 Le fasi di simulazione e codifica dell'algoritmo

Una volta sviluppato l'algoritmo, procedendo nel processo di formalizzazione si arriva alla **simulazione** dell'algoritmo, fase in cui si controlla se la sequenza di azioni ricavate è funzionante. Esistono vari metodi per simulare un algoritmo ma tutti, in pratica, consistono nell'*eseguire virtualmente le azioni dell'algoritmo* secondo la sequenza data, supponendo di avere dei dati in ingresso su cui basare la simulazione. Al termine della simulazione, controllando se il risultato ottenuto è proprio quello che ci si aspettava, è possibile capire se l'algoritmo è funzionante oppure se ci sono errori e dove sono eventualmente localizzati. In caso di errori, si dovrà ritornare alla fase precedente e rivedere l'algoritmo per eliminare eventuali malfunzionamenti. Durante la simulazione è inoltre possibile evidenziare se l'algoritmo presenta degli inconvenienti o se può essere ulteriormente migliorato, se ha ancora bisogno di qualche altra istruzione aggiuntiva per eventuali casi non considerati o altro ancora.

Prendiamo come esempio l'algoritmo associato al problema:

Calcolare la media aritmetica di 3 numeri in ingresso

e vediamone la simulazione passo per passo, utilizzando una tabella con tante colonne quante sono le variabili che sono manipolate dalle istruzioni dell'algoritmo, più una colonna che serve per indicare le azioni di cui si simula l'esecuzione (**tab. 2**)

Tabella 2

Azione	A	B	C	D
1. Inizio				
2. Ricevi in ingresso da tastiera i valori delle variabili *A*, *B*, *C*	Input: 10	Input: 7	Input: 16	
3. *Media* $\leftarrow \frac{A+B+C}{3}$				11
4. Stampa a video il valore della variabile *Media*				Output: 11
5. Fine				

Consideriamo ora l'algoritmo associato al problema:

Forniti in ingresso al programma la base e l'altezza di un triangolo, calcolarne l'area

e vediamone la simulazione passo per passo, utilizzando anche in questo caso una tabella con tante colonne quante sono le variabili manipolate dalle istruzioni dell'algoritmo, più una che indica le azioni (**tab. 3**).

Tabella 3

Azione	BaseTriangolo	AltezzaTriangolo	AreaTriangolo
1. Inizio			
2. Ricevi in ingresso da tastiera il valore della variabile *BaseTriangolo* (vincolo: maggiore di 0)	Input: 15		
3. Ricevi in ingresso da tastiera il valore della variabile *AltezzaTriangolo* (vincolo: maggiore di 0)		Input: 10	
4. $AreaTriangolo \leftarrow \frac{BaseTriangolo * AltezzaTriangolo}{2}$			75
5. Stampa a video il valore della variabile *AreaTriangolo*			Output: 75
6. Fine			

Quando si è sicuri del funzionamento dell'algoritmo, si può passare alla fase della sua **codifica** in un linguaggio di programmazione che l'elaboratore è in grado di interpretare e comprendere. La codifica consiste quindi nella *traduzione* dell'algoritmo in un insieme di istruzioni equivalenti *codificate* in un determinato linguaggio di programmazione, ottenendo finalmente il programma, che a questo punto potrà essere "caricato" nella memoria del computer per essere successivamente eseguito. Alcuni tra i principali linguaggi di programmazione sono il Pascal, il Visual Basic, il linguaggio C , il C++, Java, Python.
Vediamo come esempio la codifica prima in linguaggio Pascal e poi in linguaggio C del programma:

Forniti in ingresso al programma la base e l'altezza di un triangolo, calcolarne l'area.

Linguaggio Pascal

```
program Area_del_Triangolo;
var BaseTriangolo,AltezzaTriangolo,AreaTriangolo: real;
begin
    repeat
        write('Inserisci la lunghezza della base del triangolo: ');
        readln(BaseTriangolo);
    until BaseTriangolo>0;
    repeat
        write('Inserisci la lunghezza della altezza del triangolo: ');
        readln(AltezzaTriangolo);
    until AltezzaTriangolo>0;
    AreaTriangolo:=(BaseTriangolo*AltezzaTriangolo)/2;
    writeln('Area del triangolo: ', AreaTriangolo:3:2);
end.
```

Linguaggio C

```
#include<stdio.h>

int main()
{
    float BaseTriangolo,AltezzaTriangolo,AreaTriangolo;

    do    {
          printf("Inserisci la lunghezza della base del triangolo: ");
          scanf("%f",&BaseTriangolo);
          }
    while(BaseTriangolo<=0);
    do    {
          printf("Inserisci la lunghezza dell'altezza del triangolo: ");
          scanf("%f",&AltezzaTriangolo);
          }
    while(AltezzaTriangolo<=0);
    AreaTriangolo=(BaseTriangolo*AltezzaTriangolo)/2;
    printf("Area del triangolo: %3.2f",AreaTriangolo);
    return 0;
}
```

A questo punto, il gioco è fatto. Abbiamo raggiunto lo scopo di passare dal problema al programma risolutore corrispondente attraverso le fasi principali che costituiscono il processo di formalizzazione.
Ottenuto il programma, dovremo editarlo e caricarlo nella memoria dell'elaboratore, cioè digitarlo a computer e salvarlo con un apposito programma chiamato per l'appunto *Editor*, per poi poterlo lanciare in esecuzione in modo che il microprocessore dell'elaboratore su cui si sta lavorando ne possa eseguire le istruzioni, chiedendo in input i valori opportuni e fornendo in output i risultati richiesti.
Naturalmente molto resta ancora da vedere e ce ne renderemo conto con l'aumentare delle difficoltà dei problemi presi in considerazione, ma un primo passo verso il mondo della programmazione è ormai compiuto.

Qualcosa in più — Quali sono i linguaggi di programmazione più diffusi?

TIOBE è una azienda che si occupa in particolare di problemi legati alla qualità del codice e tra le altre attività tiene aggiornata una statistica sui linguaggi di programmazione più diffusi. La classifica è aggiornata mensilmente e nella tabella seguente è riportata la classifica di settembre 2013:

Position Sep 2013	Delta in Position	Programming Language	Ratings Sep 2013
1	—	C	16.975%
2	—	Java	16.154%
3	↑	C++	8.664%
4	↓	Objective-C	8.561%
5	↑	PHP	6.430%
6	↓	C#	5.564%
7	—	(Visual) Basic	4.837%
8	—	Python	3.169%
9	↑↑	JavaScript	2.015%
10	↑↑↑↑	Transact-SQL	2.015%

6 *Scratch*: la codifica per gioco

Nel paragrafo precedente abbiamo parlato della fase di codifica dedicata alla scrittura del codice che traduce un algoritmo nel programma corrispondente, attraverso l'uso delle istruzioni di un linguaggio di programmazione. Da quando è nata l'informatica, sono stati sviluppati tanti linguaggi di programmazione ma pochi sono davvero semplici da utilizzarsi o, comunque, sanno coniugare ricchezza elaborativa con facilità d'uso.

Per ovviare a questo inconveniente e per rendere più facile la comprensione e, quindi, la fruizione di un linguaggio di programmazione, nel 2007 un gruppo di ricercatori dell'Università M.I.T. (*Massachussets Institute of Technology*) di Boston ha studiato un modo semplice e divertente per imparare a programmare che ha portato allo sviluppo di un innovativo linguaggio di programmazione chiamato **Scratch**.

Prima di vederne il funzionamento nel dettaglio, consideriamo il seguente esempio: supponiamo di voler sviluppare un programma per scrivere sul monitor dell'elaboratore il saluto "Hello world!".

Nella **figura 12** è riportata la codifica in linguaggio C++ (e il corrispettivo in linguaggio macchina, cioè in puro codice binario) del programma che visualizza a video il risultato di **figura 13**.

```
class Hello
{
    public static void main(String [] argv)
    {
        System.out.println("Hello world!");
    }
}
```

```
11001010 11111110 10111010 10111110 00000000 00000011 00000000 00101101
00000000 00100000 00001000 00000000 00010100 00000111 00000000 00010011
00000111 00000000 00011010 00000111 00000000 00011011 00000111 00000000
00011100 00001010 00000000 00000100 00000000 00001001 00001001 00000000
00000101 00000000 00001010 00001010 00000000 00000011 00000000 00001011
00001100 00000000 00001111 00000000 00001100 00001100 00000000 00011110
00000000 00010110 00001100 00000000 00011111 00000000 00001101 00000001
00000000 00000011 00101000 00101001 01010110 00000001 00000000 00010101
00101000 01001100 01101010 01100001 01110110 01100001 00101111 01101100
01100001 01101110 01100111 00101111 01010011 01110100 01110010 01101001
01101110 01100111 00111011 00101001 01010110 00000001 00000000 00010110
00101000 01011011 01001100 01101010 01100001 01110110 01100001 00101111
01101100 01100001 01101110 01100111 00101111 01010011 01110100 01110010
01101001 01101110 01100111 00111011 00101001 01010110 00000001 00000000
00000110 00111100 01101001 01101110 01101001 01110100 00111110 00000001
00000000 00000100 01000011 01101111 01100100 01100101 00000001 00000000
00001101 01000011 01101111 01101110 01110011 01110100 01100001 01101110
01110100 01010110 01100001 01101100 01110101 01100101 00000001 00000000
00001010 01000101 01111000 01100011 01100101 01110000 01110100 01101001
01101111 01101110 01110011 00000001 00000000 00000101 01001000 01100101
[...]
```

Figura 12
La codifica in linguaggio C++ e in linguaggio macchina di un programma che stampa a video "Hello world!".

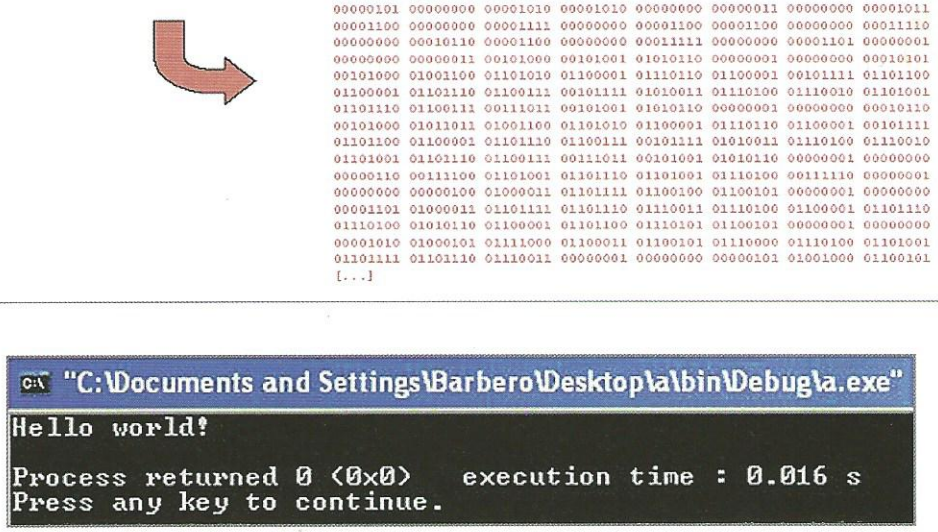


Figura 13
Il risultato a video del programma di figura 12.

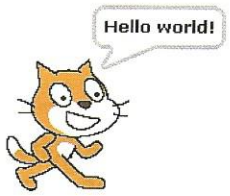
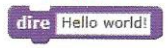


Figura 14
Quando si fa clic con il mouse sul pulsante con la bandierina verde, il gatto deve dire "Hello world!".

Vediamo quali sono invece le azioni da eseguire in *Scratch* per ottenere un risultato equivalente, mostrato in **figura 14**:

- impostare un blocco della categoria **Situazioni** del tipo *quando si clicca su* associato alla figurina della bandierina verde trascinandolo nell'area centrale dell'ambiente di sviluppo di *Scratch* 

;
- impilare un blocco della categoria **Aspetto** del tipo *dire* scrivendo nella casella di testo le parole "Hello world!" 

;
- fare clic sulla bandiera verde  che si trova in alto a destra dell'ambiente di sviluppo di *Scratch* dove compare il gatto.

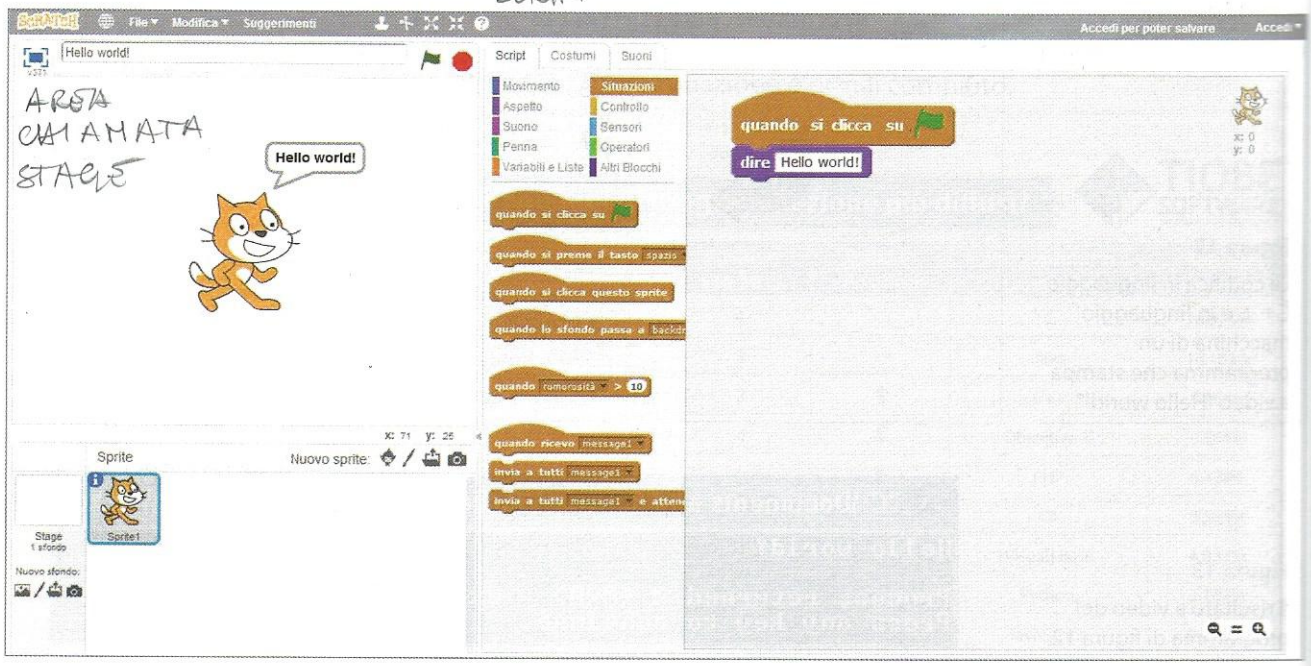
Come si può facilmente intuire, *lavorare con Scratch risulta molto più semplice e veloce* e la comprensione della codifica del programma comporta minori difficoltà. *Scratch* è un ambiente grafico di lavoro (gratuito sul sito **http://scratch.mit.edu**) per imparare la programmazione in modo semplice e creare storie interattive, giochi, animazioni grafiche, simulazioni e altro ancora, per poi condividere i programmi realizzati attraverso il Web.

La codifica dei programmi in *Scratch* consiste nell'impilare blocchi grafici, che presentano forma e colore dipendenti dall'istruzione che si vuole utilizzare, come si fa con i mattoncini delle costruzioni, pezzo dopo pezzo. Così facendo, è possibile avvicinarsi alla programmazione e capire la logica degli algoritmi in modo divertente e creativo.

I programmi di *Scratch* agiscono su oggetti grafici, disegni, immagini chiamati **sprite**, come la figura del gatto che abbiamo considerato nell'esempio precedente.

È possibile disegnare gli sprite a piacere attraverso un semplice programma di disegno, così come è possibile importare un'immagine o una foto scattata con una

Figura 15
L'ambiente *Scratch 2.0*.



macchina fotografica digitale o con la webcam. Gli sprite, inoltre, possono essere personalizzati associando costumi diversi, in modo da animarli dando loro la forma che più interessa e suoni diversi. A ogni sprite sono associate delle istruzioni che indicano che cosa deve fare: parlare, muoversi, suonare, eseguire calcoli e tanto altro ancora.
Nello screenshot di **figura 15** è possibile vedere l'esecuzione del programma *Hello world!*.

Nell'area a sinistra, chiamata **stage**, compare lo sprite del gatto, mentre nell'area centrale, chiamata area **script**, compaiono i due blocchi che corrispondono alle istruzioni che permettono allo sprite (il gatto) di interagire, pronunciando la frase indicata nel blocco di colore viola, quando viene premuta la bandierina verde.
I due blocchi utilizzati fanno parte della collezione di **blocchi** che si trovano nell'area centrale dell'ambiente di sviluppo (suddivisi in 10 categorie di altrettanti colori) e che per essere usati devono essere selezionati e trascinati nell'area di script, per poi essere eventualmente personalizzati (**fig. 16**).

Figura 16
Il trascinamento di un blocco nell'area di script.

Qualcosa in più

Mitchel Resnick, ideatore e creatore di *Scratch*.

Sono migliaia in tutto il mondo le persone che formano la comunità on line che utilizza Scratch e condivide sul sito **http://scratch.mit.edu** i propri lavori.
Ogni giorno si calcola che vengano pubblicati circa 1500 nuovi progetti che possono poi essere scaricati in locale per essere modificati e personalizzati.
Il software di Scratch è completamente gratuito e può essere utilizzato andando direttamente sul sito dove si trovano anche manuali d'uso testuali e video, tutorial e demo, e più di 4 milioni di progetti già realizzati da utenti di tutto il mondo e messi a disposizione on line.
Esiste anche la versione di *Scratch* scaricabile gratuitamente per essere utilizzata offline, sia per Mac sia per Windows e per alcune versioni di Linux. Il download può essere effettuato al link **http://scratch.mit.edu/scratch2download/**.

Figura 17

È possibile far interagire lo sprite con l'utente chiedendogli, per esempio, il nome in ingresso e personalizzando così il saluto.
Nella **figura 17** sono rappresentati i blocchi utilizzati il cui colore indica la categoria da cui sono stati selezionati.
Esaminiamo ora lo sprite in azione, passo dopo passo:

1 dopo aver fatto clic sulla bandierina verde, posta sopra lo stage, lo sprite chiede in ingresso all'utente il suo nome e quindi attende la risposta (**fig. 18**);

Figura 18

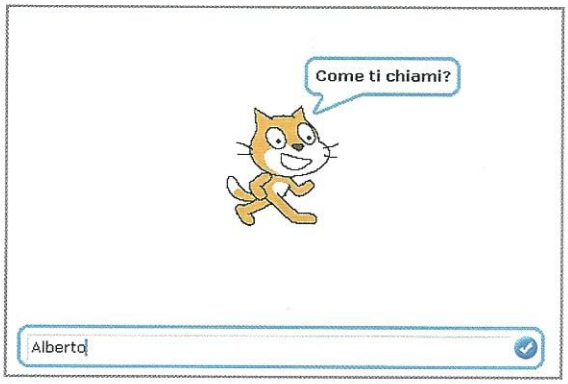


Figura 19

2 l'utente inserisce il suo nome e fa clic sul pulsante con la spunta a destra della casella di testo per confermare l'input (**fig. 19**);

Figura 20

3 a questo punto il programma, attraverso l'esecuzione del blocco di colore viola, farà "dire" allo sprite la concatenazione della parola *Ciao* con la risposta ricevuta in input dall'utente, cioè il suo nome (**fig. 20**);

Proviamo adesso a far muovere il gatto da una parte all'altra dello stage; i blocchi utilizzati sono mostrati in **figura 21**.

Figura 21

1 Lo sprite dapprima ci ricorda per 2 secondi che sta per spostarsi verso la destra dello stage di 100 passi (fig. 22);

Figura 22

Figura 23

2 effettuato lo spostamento, dopo aver invertito direzione, ci dice, sempre per 2 secondi, che sta per spostarsi indietro di 100 passi verso sinistra (fig. 23);

Se vogliamo che lo spostamento sia ripetuto più volte, come se il nostro amico gatto stesse facendo una specie di ballo "destra-sinistra", introduciamo il codice in un blocco **per sempre** (fig. 24), di colore giallo, che permette di ripetere ciclicamente delle azioni.

Figura 24

Figura 25

Vedremo lo sprite avanzare verso destra dello stage e indietreggiare, avanzare di nuovo verso destra e indietreggiare sino a quando non verrà premuto il pulsante con il segnale rosso, a destra di quello con la bandierina verde, per fermare l'esecuzione ciclica dello script.

Si potrebbe poi animare ulteriormente la figurina del gatto, simularne cioè la camminata, utilizzando i **costumi** associati allo sprite che permettono di modificarne l'aspetto, e non la forma, in modo da realizzare animazioni come nei cartoni animati, sovrapponendo appunto costumi diversi. Lo sprite del gatto presenta due costumi già disegnati (fig. 25) che, opportunamente programmati, animano il gatto facendolo muovere come se stesse camminando (fig. 26).

Figura 26

Il blocco *passa al costume seguente* permette il cambio di costume mentre i blocchi *porta stile rotazione a sinistra-destra* e *rimbalza quando tocchi il bordo* permettono di rendere corretta l'animazione. Infine, il blocco *produci suono*, di colore lilla, permette di riprodurre il miagolio del gatto…

Attraverso i blocchi della categoria **Movimento** (di colore blu), *Scratch* permette animazioni, rotazioni, scivolamenti, rimbalzi, su uno o più sprite anche contemporaneamente (come peraltro succede nei videogiochi). Con i blocchi della categoria **Aspetto** (di colore viola), *Scratch* permette anche di cambiare dinamicamente l'aspetto dello sprite, come possiamo vedere in questo semplice esempio (**fig. 27**):

Figura 27

In questo programma si agisce sullo sprite cambiandone il colore (sulla sua scala dei colori è impostato un parametro di variazione uguale a 50) e aumentandone del 30% la dimensione; dopo 2 secondi si ripete il tutto sino a quando è premuto il pulsante rosso. Il risultato che si ottiene è sorprendente (**fig. 28**).

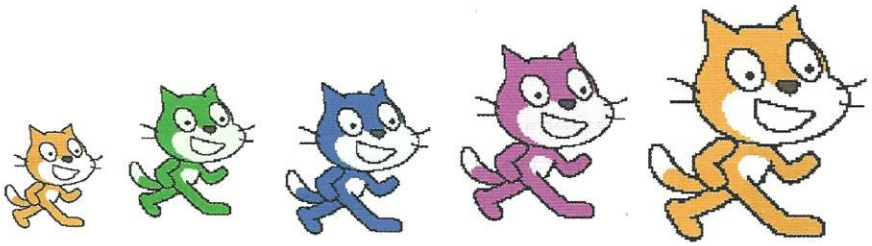


Figura 28

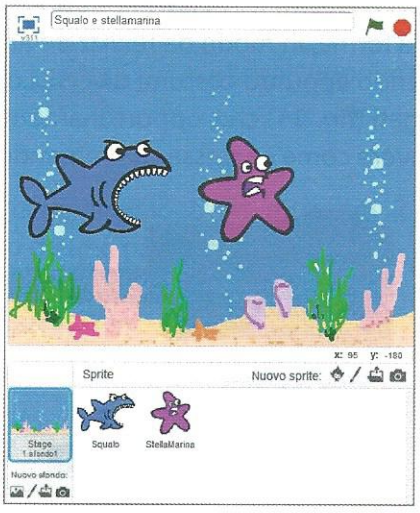


Figura 29
Anche lo stage presenta il disegno di un fondale marino come sfondo.

Proviamo ora a usare due sprite contemporaneamente, presi dalla libreria di *Scratch* che si apre facendo clic sul pulsante **Scegli uno sprite dalla libreria**, posta sotto lo stage, e sviluppiamo un programma con uno squalo che insegue una stella marina in fuga (**fig. 29**).

Lo script associato ai due sprite è lo stesso per entrambi (e va quindi associato nell'area script di ciascuno di essi) e permette alle figure di spostarsi creando un divertente effetto animazione (**fig. 30**).

Figura 30

Oltre ad agire sui movimenti degli sprite e a dare loro gli effetti desiderati, giocando su un'infinita gamma di scelte, *Scratch* permette di introdurre nei programmi la gestione dei suoni attraverso l'uso dei blocchi della categoria **Suono** (di colore lilla), offrendo suoni di strumenti musicali, voci, rumori e lasciando al programmatore la possibilità di registrare voci e suoni personalizzati per rendere l'interazione ancora più completa.
Infine, con una serie di blocchi di colore verde scuro classificati nella categoria **Penna**, è possibile costruire programmi di disegno con il computer in modo semplice e intuitivo.

Figura 31

Provate a lanciare il programma (**fig. 31**) e vedrete lo sprite del gatto muoversi nelle 4 direzioni disegnando un quadrato di 200 pixel di lato (**fig. 32**).

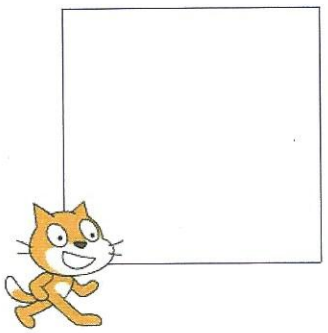


Figura 32
Il disegno di un quadrato realizzato mediante i blocchi di tipo Penna.

Oltre alle quattro categorie di blocchi di cui abbiamo parlato (Movimento, Aspetto, Suono, Penna), che contengono blocchi prettamente dedicati all'animazione e interazione degli sprite, *Scratch* offre altre categorie di blocchi che permettono di implementare i costrutti base della programmazione classica (ad esempio i cicli), la gestione degli eventi, la comunicazione tra sprite attraverso messaggi, la gestione dei sensori (ad esempio della tastiera). Vedremo l'utilizzo di questi blocchi quando avremo approfondito meglio il concetto di algoritmo e svilupperemo programmi più complessi.

7 Gli schemi di flusso

Nei paragrafi precedenti abbiamo sviluppato alcuni semplici algoritmi utilizzando frasi in linguaggio italiano per descrivere le azioni da compiere a ogni passo. Per esempio, abbiamo usato l'espressione *Salta all'azione...* per indicare il passaggio a un'istruzione non in sequenza rispetto a quella che si sta eseguendo, oppure abbiamo usato le parole *Se...allora...altrimenti* per indicare il fatto che doveva essere testata una condizione per decidere quale azione intraprendere in base al risultato del controllo. Abbiamo cioè espresso gli algoritmi facendo uso della nostra lingua e numerando le istruzioni per sapere a quale punto dell'algoritmo eventualmente "saltare". Nella **tabella 4** sono elencate le istruzioni che più frequentemente compaiono durante lo sviluppo di un algoritmo e che abbiamo già utilizzato nei precedenti esempi.

Tabella 4

Tipo di istruzione	Significato		Esempio
Azione	Istruzione che compie determinate azioni sui dati per produrre risultati intermedi o finali		*Area Triangolo* ← $\frac{BaseTriangolo * AltezzaTriangolo}{2}$ *Perimetro* ← *NumeroLati* * *Lunghezza Lato*
Controllo (Condizionale)	Istruzione che permette di prendere una decisione in base al risultato di una condizione, in modo da sapere se seguire un cammino oppure un altro		Se *NumeroLati* = 3 allora ... altrimenti ... Se $B \geq A$ allora ...
Comunicazione (Trasmissione)	Istruzione di *ingresso*: permette la comunicazione da utente a elaboratore		Ricevi in ingresso il valore della variabile *NumeroLati*
	Istruzione di *uscita*: permette la comunicazione da elaboratore a utente		Stampa a video il valore della variabile *Perimetro*
Salto	Permette di alterare l'esecuzione sequenziale delle istruzioni dell'algoritmo per saltare ad altre parti del programma	Salto *Condizionato*: associato al verificarsi di una condizione	Se *NumeroLati* = 4, allora...
		Salto *Incondizionato*: non è associato al verificarsi di una condizione	Salta all'azione 15
Inizio algoritmo	Si usa a inizio algoritmo		Inizio
Fine algoritmo	Si usa a fine algoritmo		Fine

7 Gli schemi di flusso

Nei paragrafi precedenti abbiamo sviluppato alcuni semplici algoritmi utilizzando frasi in linguaggio italiano per descrivere le azioni da compiere a ogni passo. Per esempio, abbiamo usato l'espressione *Salta all'azione...* per indicare il passaggio a un'istruzione non in sequenza rispetto a quella che si sta eseguendo, oppure abbiamo usato le parole *Se...allora...altrimenti* per indicare il fatto che doveva essere testata una condizione per decidere quale azione intraprendere in base al risultato del controllo. Abbiamo cioè espresso gli algoritmi facendo uso della nostra lingua e numerando le istruzioni per sapere a quale punto dell'algoritmo eventualmente "saltare". Nella **tabella 4** sono elencate le istruzioni che più frequentemente compaiono durante lo sviluppo di un algoritmo e che abbiamo già utilizzato nei precedenti esempi.

Tabella 4

Tipo di istruzione	Significato		Esempio
Azione	Istruzione che compie determinate azioni sui dati per produrre risultati intermedi o finali		*Area Triangolo* ← $\frac{BaseTriangolo*AltezzaTriangolo}{2}$ *Perimetro* ← *NumeroLati* * *Lunghezza Lato*
Controllo (Condizionale)	Istruzione che permette di prendere una decisione in base al risultato di una condizione, in modo da sapere se seguire un cammino oppure un altro		Se *NumeroLati* = 3 allora ... altrimenti ... Se $B \geq A$ allora ...
Comunicazione (Trasmissione)	Istruzione di *ingresso*: permette la comunicazione da utente a elaboratore		Ricevi in ingresso il valore della variabile *NumeroLati*
	Istruzione di *uscita*: permette la comunicazione da elaboratore a utente		Stampa a video il valore della variabile *Perimetro*
Salto	Permette di alterare l'esecuzione sequenziale delle istruzioni dell'algoritmo per saltare ad altre parti del programma	Salto *Condizionato*: associato al verificarsi di una condizione	Se *NumeroLati* = 4, allora...
		Salto *Incondizionato*: non è associato al verificarsi di una condizione	Salta all'azione 15
Inizio algoritmo	Si usa a inizio algoritmo		Inizio
Fine algoritmo	Si usa a fine algoritmo		Fine

Questo tipo di formalismo non è però accettabile per diverse ragioni:

- Un primo motivo riguarda proprio l'ambiguità interpretativa delle azioni e la poca standardizzazione: per esempio, uno straniero che non capisca la nostra lingua avrebbe notevoli difficoltà nel comprendere il flusso di azioni da eseguire per simulare il nostro algoritmo.
- Un altro motivo risiede nelle difficoltà d'uso di un formalismo come questo nel caso si debbano sviluppare algoritmi di grosse dimensioni con centinaia, se non migliaia, di istruzioni.
- Per tutti questi motivi, e altri ancora, occorre fare uno sforzo ulteriore per procedere con maggior correttezza nel processo di formalizzazione, introducendo un altro strumento per la rappresentazione degli algoritmi, che possa garantire uno standard sicuro ed efficace.

Si tratta del formalismo degli **schemi di flusso**, detti anche **diagrammi a blocchi** o **flow chart**: attraverso gli schemi di flusso è possibile rappresentare graficamente le istruzioni che compongono un algoritmo mediante l'utilizzo di *simboli grafici* la cui forma varia a seconda del tipo di azione che si esprime. I simboli sono uniti da frecce che rappresentano il flusso delle azioni che costituiscono l'algoritmo.
In questo modo è possibile ottenere una descrizione dell'algoritmo molto più efficace e chiara, semplice da capire e da modificare, standard e, soprattutto, senza ambiguità interpretative.
Nella **tabella 5** della pagina seguente sono raccolti i simboli associati a ogni istruzione e il significato corrispondente.

In conclusione, possiamo definire uno schema di flusso nel modo seguente:

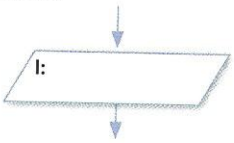
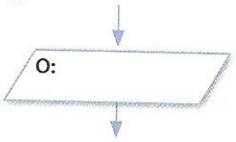
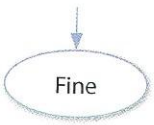
Schema di flusso

Uno *schema di flusso* (o *diagramma a blocchi* o *flow chart*) è una rappresentazione grafica di un algoritmo realizzata mediante l'utilizzo di simboli, la cui forma dipende dal tipo di azione che si vuole descrivere, uniti da frecce che rappresentano il flusso dell'esecuzione delle istruzioni che compongono l'algoritmo.

Mediante gli schemi di flusso si possono rappresentare gli algoritmi in modo più chiaro, sintetico, leggibile, comprensibile e standard.

Ricordiamo infine che, oltre a quelli già visti, per motivi di comodità e di praticità sono stati introdotti altri simboli grafici di cui eventualmente parleremo e faremo uso più avanti.

Tabella 5

Tipo di istruzione	Simbolo	Significato
Azione		*Blocco di Azione:* esegue l'azione descritta all'interno del rettangolo
Controllo (Condizionale)		*Blocco di Controllo:* verifica la condizione e se il risultato è vero passa a eseguire le istruzioni sul ramo corrispondente a vero altrimenti passa a eseguire le istruzioni sul ramo corrispondente a falso
Comunicazione (Trasmissione)	**di ingresso** 	*Blocco di Input dati:* chiede in ingresso all'utente un valore che verrà scritto in una variabile in memoria
	di uscita 	*Blocco di Output dati:* fornisce in uscita all'utente un valore che verrà visualizzato a video
Salto	È rappresentato da una freccia che si innesta in un'altra freccia nel punto dell'algoritmo cui si deve saltare 	*Salto condizionato o incondizionato:* vado a eseguire l'istruzione indirizzata dal flusso delle frecce
Inizio algoritmo		*Blocco di Inizio algoritmo*
Fine algoritmo		*Blocco di Fine algoritmo*

8 Primi esempi di schemi di flusso

Riprendiamo ora gli stessi esempi visti in precedenza e rappresentiamoli mediante schemi di flusso. Ricordiamo che il processo di formalizzazione per il passaggio da problema a programma prevede una prima fase di analisi del problema che precede quella di sviluppo dell'algoritmo.

Problema 1

Calcola la media aritmetica di 3 numeri in ingresso.

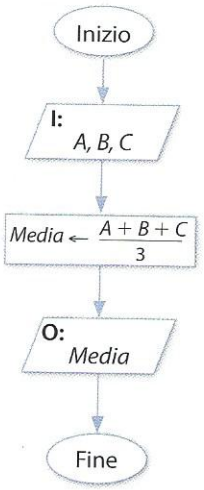
ANALISI

Dati in input *A, B, C*

Dati in output *Media*

Relazione tra I/O $Media = \frac{A + B + C}{3}$

ALGORITMO



Problema 2

Forniti in ingresso al programma la base e l'altezza di un triangolo, calcolarne l'area.

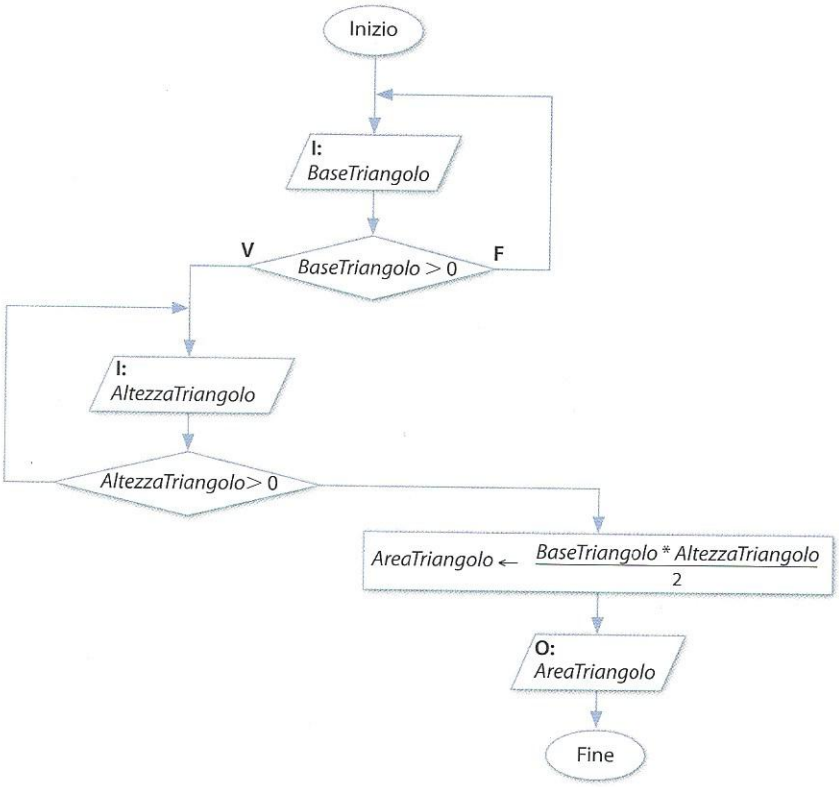
ANALISI

Dati in input *BaseTriangolo* (con *BaseTriangolo* > 0)
AltezzaTriangolo (con *AltezzaTriangolo* >0)

Dati in output *AreaTriangolo*

Relazione tra I/O $AreaTriangolo = \frac{BaseTriangolo * AltezzaTriangolo}{2}$

ALGORITMO



Nel **problema 2**, i vincoli posti sul valore delle variabili *BaseTriangolo* (>0) e *AltezzaTriangolo* (>0) vengono verificati grazie a un *ciclo*; tratteremo questa particolare istruzione più avanti.

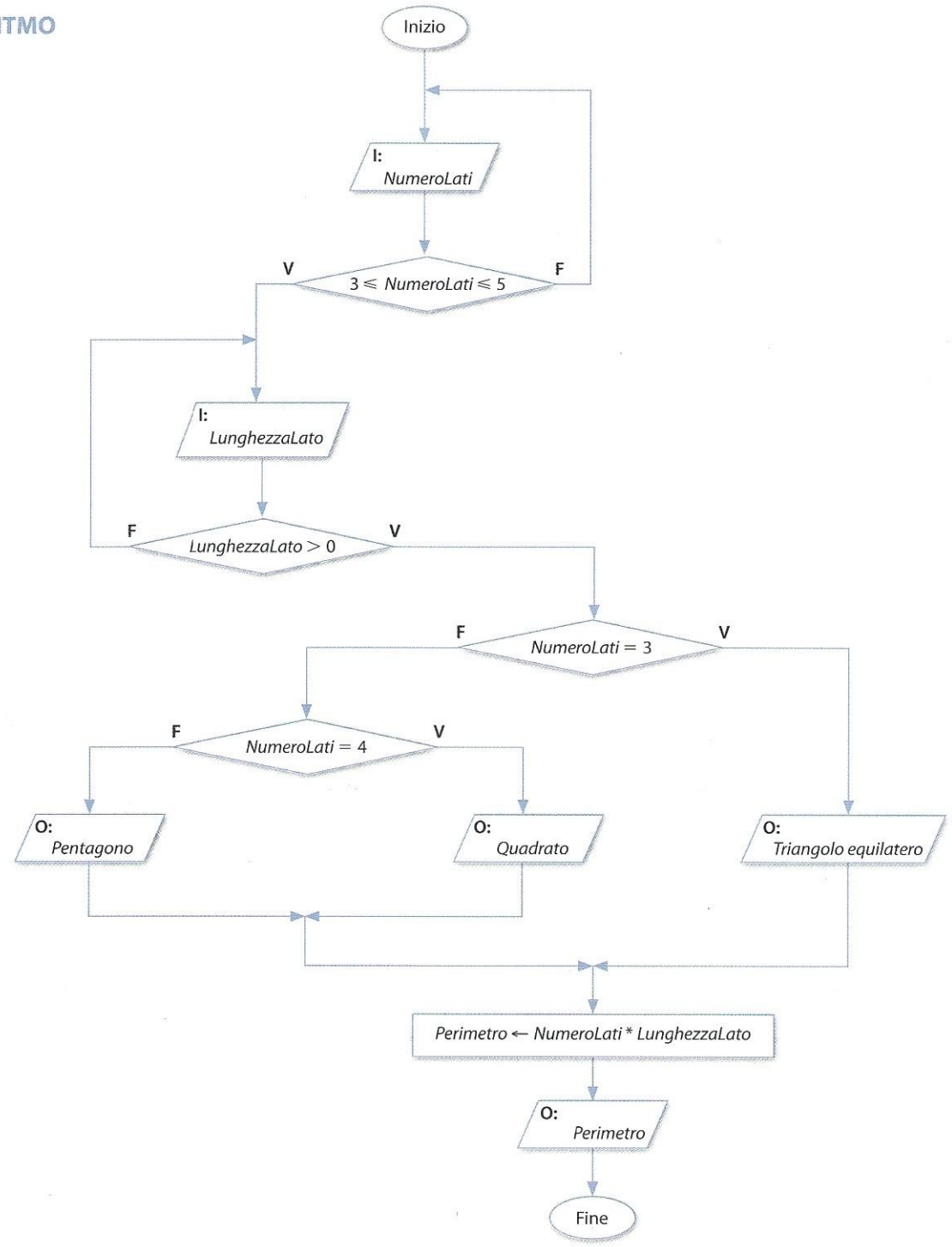
Problema 3

Fornito in ingresso al programma il numero di lati (si suppone compreso tra 3 e 5) di un poligono regolare e la lunghezza del lato del poligono, indicare di quale tipo di poligono si tratta e calcolarne il perimetro.

ANALISI

Dati in input	*NumeroLati* (con $3 \leq NumeroLati \leq 5$) *LunghezzaLato* (con *LunghezzaLato* > 0)
Dati in output	"Triangolo equilatero" o "Quadrato" o "Pentagono" *Perimetro*
Relazione tra I/O	– Se *NumeroLati* = 3 allora "Triangolo equilatero"; – se *NumeroLati* = 4 allora "Quadrato"; – se *NumeroLati* = 5 allora "Pentagono". *Perimetro* = *NumeroLati* * *LunghezzaLato*

ALGORITMO

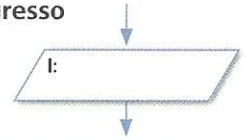
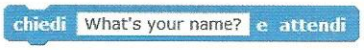
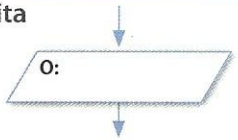
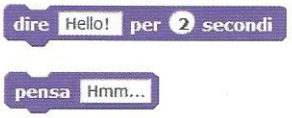
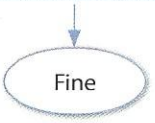


Anche nel **problema 3**, i vincoli posti sul valore delle variabili *NumeroLati* (compreso tra 3 e 5) e *LunghezzaLato* (>0) sono verificati grazie a un ciclo.

9 Dai simboli degli schemi di flusso ai blocchi di *Scratch*

Dovendo codificare un algoritmo mediante il linguaggio di programmazione *Scratch*, prendiamo nuovamente in considerazione la tabella 5 con l'elenco dei simboli utilizzati per lo sviluppo degli schemi di flusso, associando a ciascuno di essi alcuni fra i blocchi corrispondenti di *Scratch* (tab. 6).

Tabella 6

Tipo di istruzione	Simbolo	Blocco *Scratch*
Azione		
Controllo (Condizionale)		
Comunicazione (Trasmissione)	**di ingresso** 	
	di uscita 	
Salto		Tratteremo questi blocchi solo successivamente, quando parleremo di *cicli*
Inizio algoritmo		
Fine algoritmo		

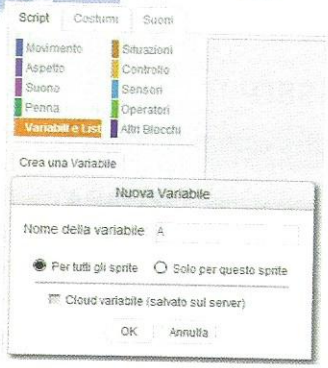


Figura 33

Figura 34

Figura 35

Come per gli altri linguaggi di programmazione, anche *Scratch* permette di gestire le variabili e di elaborarle. A questo scopo, *Scratch* fornisce nella categoria **Variabili e liste** (di colore arancione), dei blocchi che servono a memorizzare ed elaborare dati del programma. A differenza degli altri blocchi, le *variabili vanno prima create* associando loro un identificatore. Facendo clic sul pulsante **Nuova variabile** (**fig. 33**) è possibile creare le variabili che ci servono nella codifica dell'algoritmo. In questo caso è stata creata la variabile identificata da *A*, ottenendo il risultato di **figura 34**.
Oltre alla variabile *A* appena creata, compaiono nell'area quattro nuovi blocchi di colore arancione che permettono di gestire le operazioni di *assegnazione*, come si vede negli esempi che seguono.

Dovendo eseguire:

***A* ← 1**

- si trascina nell'area **Script** il blocco porta A a 0 ;
- si modifica poi lo 0 con il valore 1 porta A a 1 ottenendo l'istruzione di assegnazione del valore 1 alla variabile *A*.

***A* ← *A* + 1**

- si trascina il blocco cambia A di 1 nell'area **Script** ottenendo l'istruzione di incremento di 1 della variabile *A*.

Supponendo di lavorare con 2 variabili chiamate rispettivamente *A* e *B*, si dovrà prima creare la seconda variabile ottenendo la situazione indicata in **figura 35**.
A questo punto, dovendo eseguire:

B* ← *A

- si trascina il blocco porta A a 0 nell'area **Script**;
- si seleziona poi dalla casella in discesa in corrispondenza al triangolino nero porta A a 0 (A, B) l'identificatore della variabile *B*;
- si trascina nella casella bianca, dove c'è il valore 0, il blocco corrispondente alla variabile *A* (A) ottenendo infine porta B a A , cioè l'assegnazione della variabile *A* alla variabile *B*.

A* ← *B* – *A

- si trascina il blocco porta A a 0 nell'area **Script**;
- si trascina nella casella bianca, in sostituzione del valore 0, il blocco (-) che rappresenta l'operazione di differenza, preso dalla categoria degli **Operatori** (di colore verde chiaro);
- si spostano nelle due caselle bianche i blocchi corrispondenti alle variabili *A* (A) e *B* (B) ottenendo infine porta A a (B - A) , cioè l'assegnazione di *B* – *A* alla variabile *A*.

Per effettuare lo scambio di *A* con *B* (mediante l'utilizzo di una variabile ausiliaria *Aus*), eseguiamo i tre passaggi:

Aus* ← *A ***A* ← *B*** ***B* ← *Aus***

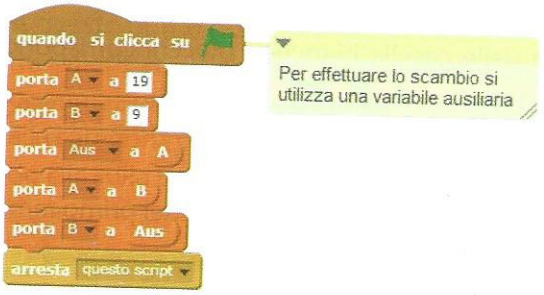
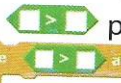
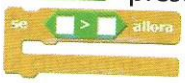
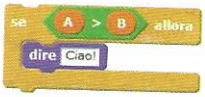


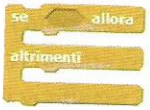
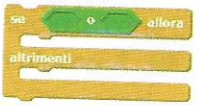
Figura 36
Nell'esempio si è inserito un commento al codice facendo clic con il tasto destro del mouse nell'area script.

Supponendo di aver creato in *Scratch* anche la variabile *Aus* e di assegnare in partenza il valore 19 alla variabile *A* e 9 alla variabile *B*, si avrà il codice di **figura 36**.

Le variabili sono anche utilizzate come operandi all'interno delle condizioni che si trovano nei blocchi di controllo. Per codificare una condizione (**fig. 37**), *Scratch* presenta nella categoria degli **Operatori**, di colore verde chiaro, dei blocchi che permettono di rappresentare gli operatori usati nei controlli.
Per costruire la condizione rappresentata in **figura 37a**:

- si usa il blocco **se... allora** di color giallo della categoria **Controllo**  ;
- nell'esagono si trascina poi il blocco  preso dalla categoria **Operatori** contenente il simbolo > ottenendo  ;
- si spostano nelle due caselle bianche i blocchi corrispondenti alle variabili *A* e *B* in modo da ottenere  ;
- si incastra infine il blocco  all'interno del blocco **se... allora**, ottenendo infine  .

Per codificare la condizione rappresentata in **figura 37b**:

- si seleziona il blocco **se... allora... altrimenti**  di color giallo della categoria **Controllo**;
- nell'esagono si trascina poi il blocco  preso dalla categoria **Operatori** e corrispondente all'operatore booleano OR per combinare le due condizioni sulle variabili *A* e *B* in modo da avere  ;

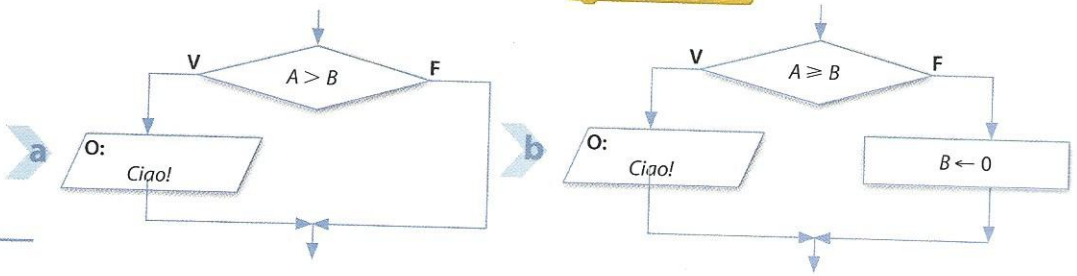
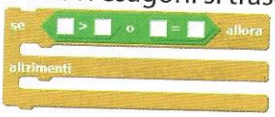
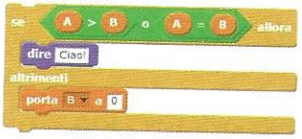


Figura 37

- nei due nuovi esagoni si trascinano gli operatori con i simboli > e = in modo da avere  ;
- si spostano nelle caselle bianche i blocchi corrispondenti alle variabili *A* e *B*, ottenendo quindi  ;
- si incastra il blocco  all'interno del primo ramo del blocco **se... allora... altrimenti**;
- si incastra il blocco  all'interno del secondo ramo del blocco **se... allora... altrimenti**, ottenendo infine il risultato richiesto  .

Vediamo infine un ulteriore esempio di codifica in *Scratch* di una condizione un poco più complessa:

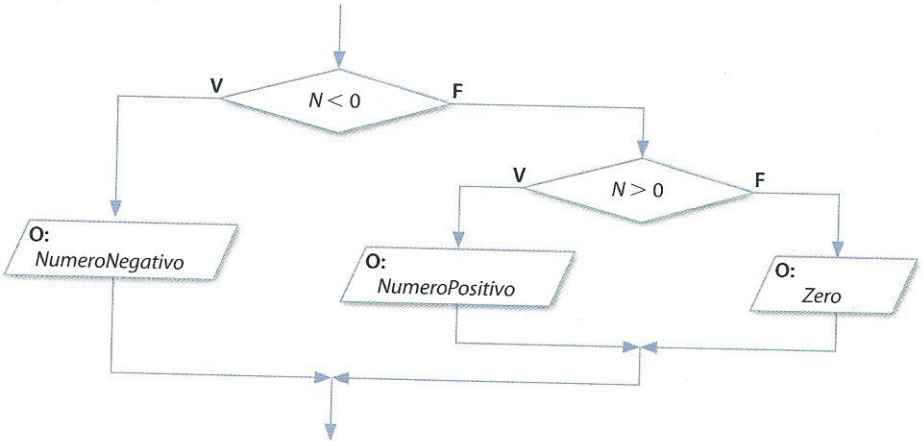


Figura 38

Per codificare le condizioni dell'esempio dovrò fare uso di blocchi di controllo *Scratch* del tipo **se... allora... altrimenti** annidati uno dentro l'altro ottenendo al termine il seguente script:

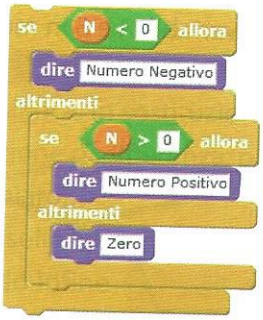


Figura 39