



AICA

Associazione Italiana per l'Informatica
ed il Calcolo Automatico



*Ministero dell'Istruzione
dell'Università e Ricerca*



OLIMPIADI ITALIANE DI INFORMATICA

Materiale didattico per la preparazione alla selezione scolastica

Edizione 2012

Versione per il linguaggio C



**Susanna Anelli
Alessandro Bugatti
Giuseppe Callegarin**

Sirmione , ottobre 2012

© 2012 Susanna Anelli, Alessandro Bugatti, Giuseppe Callegarin

Questo materiale è rilasciato sotto la disciplina della licenza Creative Commons Attribuzione – Non commerciale - Non opere derivate 3.0 Italia, il cui testo valido ai fini legali è disponibile alla pagina web:

<http://creativecommons.org/licenses/by-nc-nd/3.0/it/deed.it>

Per le parti dell'opera di pubblico dominio (i testi dei problemi), tale condizione non è in alcun modo modificata dalla licenza.



a Marta Genovì De Vita

Finalità e struttura del materiale didattico

Questo materiale didattico è stato commissionato dal Comitato Olimpico delle Olimpiadi Italiane di Informatica con lo scopo di aggiornare la precedente versione dell'ottobre 2007. L'incarico è stato affidato a tre docenti di informatica da anni impegnati nella didattica di questa disciplina e nel coinvolgimento della propria scuola nelle gare.

Nella stesura del materiale si è pensato a due tipi di utenza:

- i *docenti* delle varie scuole che, auspicabilmente, intendono organizzare attività preparatorie con cui aiutare i propri allievi ad affrontare la gara;
- gli *allievi* che intendono affrontare da autodidatti le selezioni scolastiche qualora la scuola di appartenenza non riesca ad organizzare per tempo delle iniziative di preparazione alle gare.

Le selezioni scolastiche prevedono attualmente tre diverse tipologie di problemi:

- problemi di natura logico-matematica;
- problemi di programmazione riferiti ad uno dei linguaggi C o Pascal;
- problemi di carattere algoritmico (dal 2010).

Questo materiale si rivolge alle prime due tipologie di problemi proponendo delle soluzioni commentate ad alcuni dei problemi assegnati negli anni 2008, 2009, 2010 e 2011.

I problemi di carattere algoritmico sono stati introdotti nel 2010 con lo scopo di verificare abilità di programmazione e di problem-solving indipendenti da specifici linguaggi di programmazione.

La struttura di questi problemi è ancora in via di stabilizzazione e comunque non consente al momento di predisporre del materiale didattico valido per diversi anni.

Il documento è suddiviso in tre sezioni:

- selezione di problemi logico-matematici;
- selezione di problemi di programmazione in C;
- breve guida al linguaggio C (lo stretto indispensabile per superare le selezioni scolastiche).

Questo materiale è scaricabile dalla sezione *Documentazione* del sito delle Olimpiadi di Informatica:

<http://www.olimpiadi-informatica.it/>

Esiste anche la versione che ha il Pascal come linguaggio di riferimento.

I docenti possono usare il materiale in vari modi. Una tecnica che si è rivelata efficace è quella di mettere assieme il testo di una gara selezionando un certo numero dei problemi proposti, di somministrarli e solo dopo confrontare le soluzioni ottenute con quelle commentate riportate nel documento.

Ringraziamenti

Siamo grati al Prof. *Nello Scarabottolo* del Comitato Olimpico, per i preziosi consigli volti a risolvere indecisioni sul modo di procedere. Preziosi rilievi sono arrivati anche da colleghi e allievi che si sono prestati alla prima lettura dei materiali.

Dedichiamo il frutto del nostro lavoro all'Ispezzione Ministeriale *Marta Genovè De Vita*, nonché stimatissima Presidente del Comitato Olimpico, che ci ha lasciato il 7 settembre di quest'anno. Questa era solo una delle innumerevoli iniziative che ha avviato a sostegno dell'educazione scientifica nelle scuole italiane. Ci rimarrà sempre il rammarico di non aver potuto condividere con Marta il momento della presentazione/diffusione di questo materiale ma anche l'onore e la fortuna di aver discusso con Lei le linee da seguire, come sempre ispirate a rigore, buon senso e mediazione.

S. Anelli – A. Bugatti – G. Callegarin

Sirmione, Ottobre 2012

Selezione di problemi logico matematici con soluzioni commentate

Premessa

Il tipo di problemi logico-matematici proposti alle selezioni scolastiche delle Olimpiadi di Informatica si possono ritrovare anche in molti tipi di gare scientifiche (in particolare di Matematica e di Fisica). Le conoscenze di matematica e di logica richieste vengono di solito coperte dai corsi di matematica del biennio di ogni tipo di scuola superiore. L'obiettivo non è tanto quello di verificare delle conoscenze specifiche ma di misurare la capacità di combinare *intuito* e strumenti di ragionamento logico-matematico.

Per questo motivo, in questa selezione dei problemi assegnati nelle edizioni scolastiche delle Olimpiadi di Informatica degli anni 2008, 2009, 2010 e 2011 le soluzioni commentate prevedono spesso delle versioni "intuitive" o comunque alternative a quelle "formali", senza ricorrere alle quali sarebbe impossibile rientrare nel tempo concesso per risolvere tutti i quesiti.

Ogni problema è caratterizzato da:

- un codice identificativo nella forma **LM** *progressivo.anno.progressivoilInterno.difficoltà*;
- una categoria entro cui si può classificare il problema;
- un elenco di parole chiave che identificano gli argomenti coinvolti nella soluzione del problema.

I problemi sono stati proposti con il seguente ordine di categorie e a vari livelli di difficoltà:

- problemi di *deduzione logica*;
- problemi di *ottimizzazione*;
- problemi di *calcolo*;
- problemi di *insiemistica e calcolo combinatorio*.

Si consiglia di provare a risolvere il problema prima di leggere la soluzione commentata che comprende, in molti casi, più alternative significative. Si potrà anche vedere che, in alcuni casi, le soluzioni più veloci passano attraverso la conoscenza di nozioni elementari di calcolo combinatorio o sui numeri naturali (come la somma dei naturali da 1 ad n).

In alcuni casi si propone come utile esercizio qualche problema simile assegnato anche negli anni precedenti.

Per alcuni problemi sono stati rivisti i livelli di difficoltà attribuiti nel testo della gara con il sistema dei punti.

Problema: LM 1.2008.1.F

Difficoltà: facile

Classificazione: problemi di deduzione logica

Parole chiave: regole di inferenza

Domanda

Si immagini che il tempo segua sempre questa semplice regola:

Se oggi piove, allora domani ci sarà il sole.

Sapendo che oggi piove, dire:

- Come era il tempo ieri?

- Che tempo farà domani?

Soluzione commentata

In questo problema occorre rendersi conto che si suppone una estrema semplificazione del tempo meteorologico: o c'è il sole o piove. Per sapere che tempo farà domani si applica banalmente la regola e quindi si ottiene che domani ci sarà il sole. Per ieri possiamo considerare che se ieri avesse piovuto allora oggi ci sarebbe stato il sole, ma invece oggi piove, quindi non è possibile che ieri abbia piovuto, di conseguenza ci deve essere stato il sole. Quindi la risposta è ieri sole, domani sole.

Problema: LM 2.2008.7.M

Difficoltà: media

Classificazione: problemi di deduzione logica

Parole chiave: regole di inferenza

Domanda

Quattro amiche si conoscono dall'inizio della scuola.

I loro nomi sono: Claudia, Daria, Laura, Maria.

Se solo una delle seguenti affermazioni è vera, chi è la più intelligente del gruppo?

Laura: "Maria è la più intelligente di tutte"

Maria: "Daria è la più intelligente di tutte"

Daria: "Non sono io la più intelligente di tutte"

Claudia: "Non sono io la più intelligente di tutte"

Risposte:

a) Claudia

b) Daria

c) Laura

d) non è possibile stabilirlo

Soluzione commentata

Usando le iniziali dei nomi rappresentiamo le affermazioni delle quattro amiche:

L	M	D	C
è M	è D	non sono io	non sono io

Possiamo raggiungere la soluzione in due modi:

Prima soluzione

Verifichiamo le implicazioni che si hanno supponendo vera una singola affermazione alla volta:

-se L fosse vera allora anche D e C sarebbero vere (incompatibile);

-se M fosse vera allora anche C sarebbe vera (incompatibile);

-se C fosse vera, allora D dovrebbe essere falsa, ma se D fosse falsa allora M sarebbe vera (incompatibile);

-se D fosse vera allora C dovrebbe essere falsa, quindi C sarebbe la più intelligente. In questo caso L e M sarebbero false.

La verità è quella di D e Claudia (C) è la più intelligente (risposta a).

Seconda soluzione

Verifichiamo quale risposta è compatibile con la condizione che solo un'affermazione è vera.

Se la risposta giusta fosse a) allora l'unica affermazione vera sarebbe quella di Daria e quindi questa è la soluzione. Come controprova proviamo anche con la b) e la c): se b) fosse corretta allora sia l'affermazione di Maria che quella di Claudia sarebbero vere, quindi è da scartare, se invece fosse c) sia l'affermazione di Daria che quella di Claudia sarebbero vere quindi anche in questo caso è da scartare.

Problema simile: 2003.6

Problema: LM 3.2009.7.F

Difficoltà: facile

Classificazione: problemi di deduzione logica

Parole chiave: regole di inferenza

Domanda

Federico ha trovato in soffitta tre scatole speciali. La nonna gli ha detto che una di quelle tre scatole è piena di giocattoli mentre le altre due sono vuote, ma purtroppo potrà aprirne una sola. Per fortuna il coperchio di ogni scatola riporta un'affermazione sul contenuto della scatola stessa:

-la scatola A riporta l'affermazione "I giocattoli non sono qui"

-la scatola B riporta l'affermazione "I giocattoli non sono qui"

-la scatola C riporta l'affermazione "I giocattoli sono nella scatola B"

La nonna svela un segreto a Federico: "una e solo una delle tre affermazioni è vera !".

Dove sono i giocattoli?

Risposte:

a) I giocattoli sono nella scatola A

b) I giocattoli sono nella scatola B

c) I giocattoli sono nella scatola C

d) È impossibile che una e solo una delle tre affermazioni sia vera

Soluzione commentata

Rappresentiamo graficamente la situazione

A	B	C
I giocattoli non sono qui	I giocattoli non sono qui	I giocattoli sono nella scatola B

A questo punto si controlla quale soluzione sia compatibile con i dati del problema e si può ragionare nei seguenti due modi.

Prima soluzione

Se l'affermazione di A fosse vera, i giochi sarebbero in B o in C.

Se fossero in B, allora sarebbe vera anche C (contro l'ipotesi che solo una è vera).

Se fossero in C, allora sarebbe vera anche B (contro l'ipotesi che solo una è vera).

L'affermazione di A è quindi falsa.

Se l'affermazione di C fosse vera, i giochi sarebbero in B, ma allora sarebbe vera anche A (contro l'ipotesi che solo una è vera).

L'affermazione di C è quindi falsa.

Quindi deve essere vera l'affermazione di B.

I giochi quindi possono essere in A o in C.

Se fossero in C, sarebbe vera anche la A (contro l'ipotesi).

Se fossero in A tutto sarebbe corretto perché le affermazioni di A e di C sarebbero false, quindi i giochi sono nella scatola A (risposta a).

Seconda soluzione

Proviamo a supporre che i giochi siano in una delle tre scatole e vediamo cosa ne consegue: se i giochi fossero in A, avremmo che l'affermazione di A sarebbe falsa, quella di B vera e quella di C falsa, il tutto compatibile con quanto svelato dalla nonna, quindi i giochi saranno in A (risposta a). Per controprova si possono verificare le altre due possibilità:

- se fossero in B l'affermazione di A sarebbe vera, quella di B sarebbe falsa e quella di C vera, violando così il segreto della nonna;
- se fossero in C l'affermazione di A sarebbe vera, quella di B sarebbe vera e quella di C falsa, violando nuovamente il segreto della nonna.

Problema simile: 2003.6

Problema: LM 4.2011.5.D

Difficoltà: difficile

Classificazione: problemi di ottimizzazione

Parole chiave: analisi dei casi possibili

Domanda

Le Ferrovie dello Stato hanno deciso di dotare ogni treno di un team di 5 persone per il rapporto a bordo con i viaggiatori. Il team deve essere composto da due persone che controllano i biglietti (Servizio Biglietti) una persona per interagire con i viaggiatori nel caso di reclami a bordo (Servizio Clienti) e due persone per il servizio d'ordine. Nell'organizzare un team siamo a conoscenza delle seguenti preferenze:

- A,B e C sono disponibili per il Servizio Biglietti
- C,D ed E sono disponibili per il Servizio Clienti
- F,G,H sono disponibili per far parte del Servizio d'Ordine
- A e H possono lavorare solo se sono insieme nello stesso team
- E preferisce lavorare solo se F lavora

Quanti diversi team è possibile organizzare rispettando le preferenze di cui ai precedenti punti da 1 a 5?

Soluzione commentata

Riassumiamo le condizioni

- A,B,C **SB** (Servizio Biglietti)
- C,D,E **SC** (Servizio Clienti)
- F,G,H **SO** (Servizio d'Ordine)
- A,H devono lavorare insieme
- E lavora se lavora F (e non il viceversa!!!)

Soluzione tabellare

Costruiamo una tabella con tutte le possibilità ed eliminiamo quelle che non rispettano i vincoli:

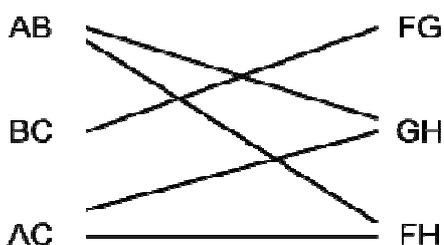
SB	SC	SO	
AB	C	FG	non ammessa: A deve essere con H
		GH	Ammessa
		FH	Ammessa
	D	FG	non ammessa: A deve essere con H
		GH	Ammessa
		FH	Ammessa
	E	FG	non ammessa: A deve essere con H
		GH	non ammessa: E lavora se lavora F
		FH	Ammessa
BC	C		non ammessa: C non può svolgere due funzioni contemporaneamente
	D	FG	Ammessa

		GH	non ammessa: H deve essere con A
		FH	non ammessa: H deve essere con A
	E	FG	Ammessa
		GH	non ammessa: H deve essere con A
		FH	non ammessa: H deve essere con A
AC	C		non ammessa: C non può svolgere due funzioni contemporaneamente
	D	FG	non ammessa: A deve essere con H
		GH	Ammessa
		FH	Ammessa
	E	FG	non ammessa: A deve essere con H
		GH	non ammessa: E lavora se lavora F
		FH	Ammessa

Come si può vedere rimangono 10 possibili team (risposta 10).

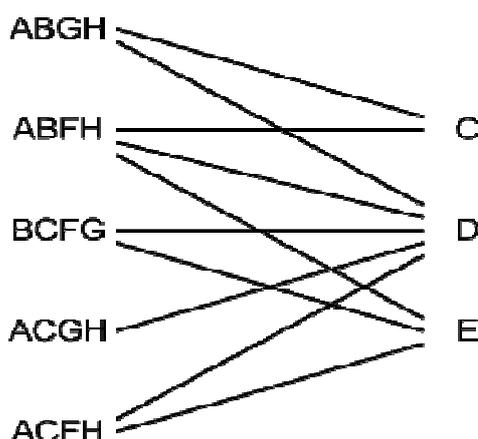
Soluzione grafica

In altro modo di procedere che può evitare di dover costruire tutta la tabella (ma non è necessariamente più veloce) è quello di usare un grafo per costruire tutte le squadre possibili. Si può partire creando i team possibili contenenti le due persone dei Biglietti e le due del Servizio d'Ordine, trascurando per il momento la persona del Servizio Clienti. Nel grafo disegniamo le possibili coppie per ogni servizio (essendo tre le persone che fanno il servizio, le coppie diverse per ognuno sono tre) e uniamo le coppie per formare i team che rispettano la regola 4 (la 5 si applicherà nel prossimo passaggio) tramite una linea: il numero di linee rappresenterà quindi i team possibili.



Dal grafo si vede quindi che sono 5 i team possibili, ma adesso bisogna aggiungere la persona del Servizio Clienti scelta tra C, D e E.

Riapplicando la stessa procedura e verificando che sia rispettata la condizione 5 (e ovviamente che C non compaia in una squadra in cui lavora già nel Servizio Biglietti) si ottiene:



Contando le linee si vede che sono proprio 10, ognuna corrispondente a un possibile team (risposta 10).

Problema: LM 5.2009.10.M

Difficoltà: media

Classificazione: problemi di ottimizzazione

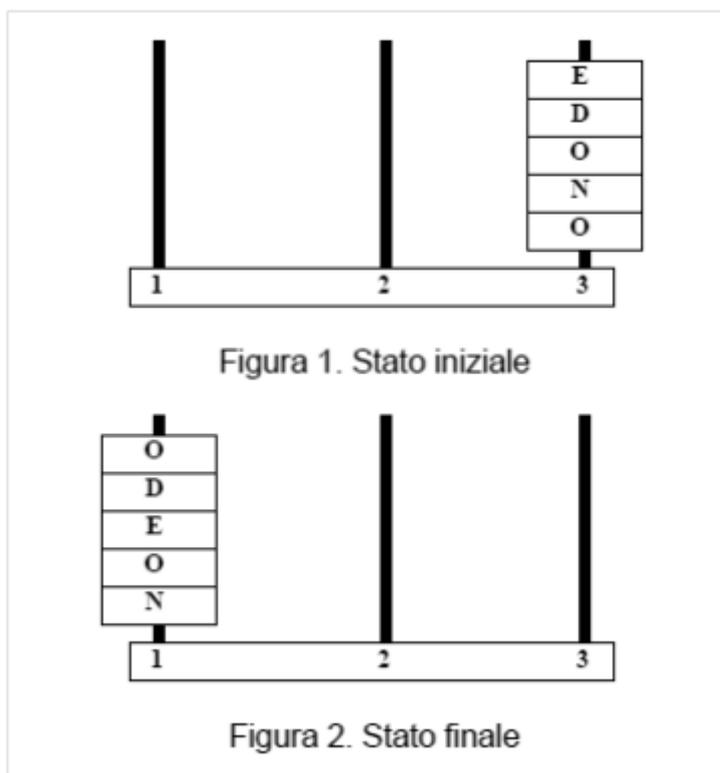
Parole chiave: analisi per tentativi

Domanda

Un gioco è realizzato inserendo in una tavoletta tre pioli numerati con 1, 2, 3 (come mostrato in figura 1). Sul piolo 3 c'è una pila di dischi, su ciascuno dei quali è incisa una lettera maiuscola in modo che dall'alto in basso si legga EDONO (come mostrato sempre in figura 1).

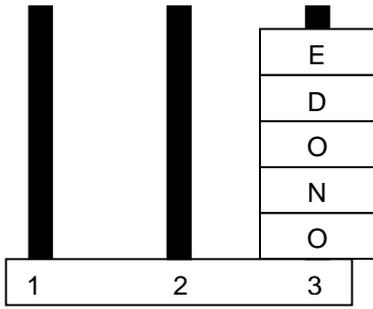
Si possono spostare i dischi prelevandoli uno alla volta dalla cima della pila di un piolo e infilandoli in un altro piolo: ciascun spostamento costituisce una mossa.

Qual è il numero minimo di mosse necessarie per trasferire i dischi al piolo 1 in modo che dall'alto in basso si legga ODEON (come mostrato in figura 2)?

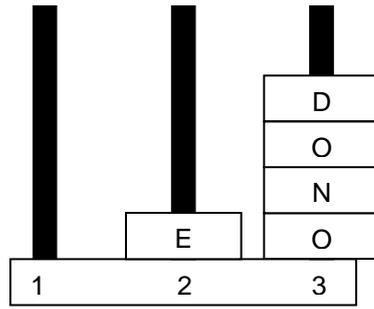


Soluzione commentata

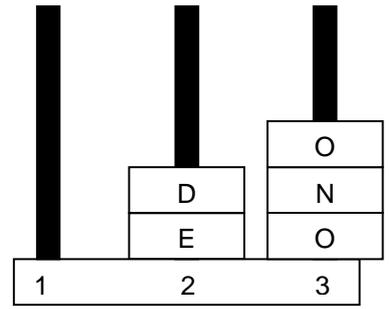
Risulta chiaro che bisogna portare il disco con la lettera N nel piolo 1. Per far questo occorre prima depositare nel piolo 2 le lettere E,D,O; poi si “naviga a vista” nello spazio degli stati possibili verso lo stato finale richiesto, come illustrato in figura.



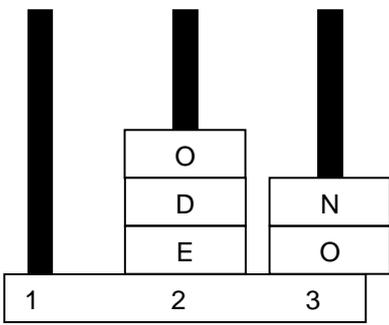
Stato iniziale



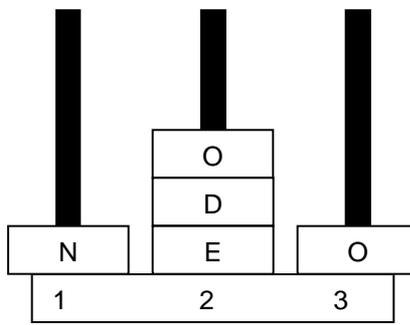
Mossa 1



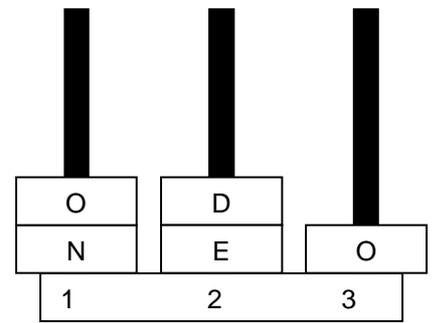
Mossa 2



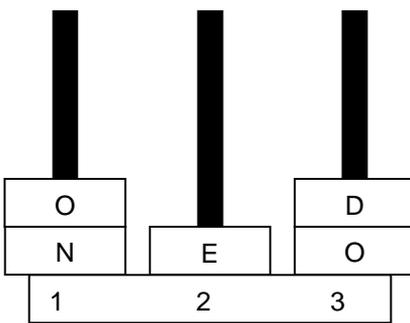
Mossa 3



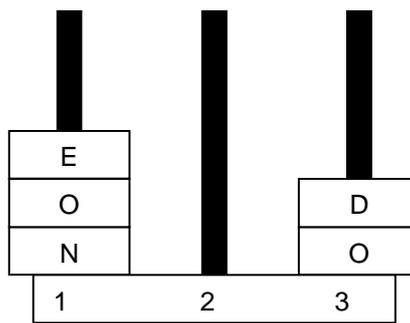
Mossa 4



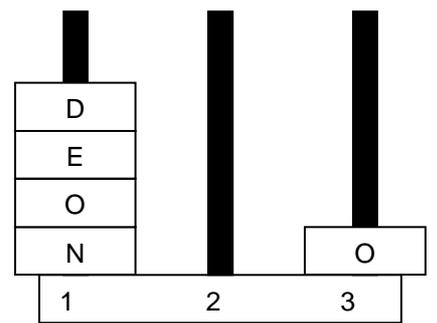
Mossa 5



Mossa 6



Mossa 7



Mossa 8

Quindi con l'ultima ovvia mossa la risposta è 9.

Problema: LM 6.2008.2.F

Difficoltà: facile

Classificazione: problemi di calcolo

Parole chiave: equazioni di primo grado, sistemi

Domanda

Aldo oggi compie gli anni e i parenti si riuniscono per festeggiarlo.

Sua zia Bruna, che non lo vede da tempo, esclama "Ma come sei diventato alto!!".

Aldo risponde: "Lo sai, zia? La mia statura, in centimetri, è dodici volte la mia età, ma tre anni fa ero alto tredici volte la mia età. Eppure sono cresciuto di 24 centimetri".

Quanti anni ha compiuto Aldo?

Soluzione commentata

Sia S la statura di Aldo adesso e E la sua età corrente. Allora 3 anni fa l'età di Aldo era $E - 3$ e la sua statura era $S - 24$. Unendo queste informazioni con i rapporti tra età e statura oggi e 3 anni fa si ottiene il seguente sistema:

$$\begin{cases} S = 12E \\ S - 24 = 13(E - 3) \end{cases}$$

che risolto porta alla soluzione $E = 15$.

Problemi simili: 2008.6

Problema: LM 7.2008.4.F

Difficoltà: facile

Classificazione: problemi di calcolo

Parole chiave: equazioni di primo grado, sistemi, sostituzioni

Domanda

Una bottiglia di vino costa 10 euro; il vino costa 9 euro in più della bottiglia vuota.

La bottiglia vuota costa:

Risposte:

a) 9 euro

b) solo 1 euro

c) 50 centesimi

d) nessuna delle precedenti

Soluzione commentata

Impostiamo un sistema di due equazioni in due incognite B (costo della bottiglia vuota) e V (costo del vino):

$$\begin{cases} B + V = 10 \\ V = B + 9 \end{cases}$$

La prima equazione rappresenta che il costo della bottiglia piena è dato dal costo della bottiglia più il costo del vino e la seconda che il vino costa 9 euro in più della bottiglia vuota

La soluzione del sistema è $B = 0,5$ (risposta c).

Problema: LM 8.2008.5.F

Difficoltà: facile

Classificazione: problemi di calcolo

Parole chiave: proporzioni, calcolo mentale

Problemi Logico:Matematici - Materiale didattico per la preparazione alle selezioni scolastiche delle OII - Ottobre 2012

Domanda

In una pasticceria tre pasticceri riempiono 12 cornetti di cioccolato in un minuto.
Se si raddoppia il personale quanti minuti occorreranno per farcire 48 cornetti?

Risposte:

- a) 1
- b) 4
- c) 6
- d) nessuna delle precedenti

Soluzione commentata

Se il numero di camerieri raddoppia, raddoppierà nello stesso modo anche la quantità di cornetti che riescono a farcire in un minuto, che quindi diventeranno 24. Ne consegue che 6 camerieri impiegheranno 2 minuti a farcire 48 cornetti, che sono il doppio di 24.

Riassumendo nella seguente tabella:

Numero pasticceri	Numero minuti	Numero cornetti
3	1	12
6	1	24
6	2	48

Come si vede le risposte a), b), c) sono errate, quindi quella corretta è la d).

Problema: LM 9.2008.6.F

Difficoltà: facile

Classificazione: problemi di calcolo

Parole chiave: equazioni di primo grado

Domanda

Un padre ha 49 anni e il figlio 27.
Quando l'età del padre è stata tripla di quella del figlio?

Risposte:

- a) 15 anni fa
- b) 11 anni fa
- c) 16 anni fa
- d) non è possibile stabilirlo

Soluzione commentata

Dobbiamo tornare indietro di x anni, fino a che l'età del padre è 3 volte l'età del figlio.

Sia P l'età del padre e F l'età del figlio in questo momento. Quindi deve essere: $P - x = 3(F - x)$

Sostituendo si ottiene $x = 16$ (risposta c).

Problema: LM 10.2008.11.M

Difficoltà: media

Classificazione: problemi di calcolo

Parole chiave: calcolo mentale, aritmetica modulare

Domanda

Uno studente, usando le dita della sua mano sinistra, inizia a contare indicando il pollice 1, l'indice 2, il medio 3, l'anulare 4, il mignolo 5, poi cambiando direzione indica di nuovo l'anulare 6, il medio 7, l'indice 8, il pollice 9, poi ancora l'indice 10, il medio 11, l'anulare 12 e così via.

Su quale dito si fermerà quando avrà raggiunto il numero 2008?

Risposte:

- a) pollice
- b) indice
- c) medio
- d) anulare

Soluzione commentata

In questo problema è sufficiente notare che dopo aver contato fino a 8 ci si ritroverà nella situazione di partenza, cioè si ripartirà dal pollice. Prendendo quindi 2008 e dividendolo per 8 il resto della divisione ci dirà a quale dito si arriva. In questo caso il resto 0 dice che siamo pronti a ripartire con il pollice, quindi la soluzione è l'indice. Se non si è convinti a causa del resto 0, basta vedere che con 2000 si arriva alla situazione di partenza e poi basta contare fino a 8 per arrivare sull'indice (risposta b).

Problema: LM 11.2010.1.F

Difficoltà: facile

Classificazione: problemi di calcolo

Parole chiave: equazioni di primo grado, sistemi, sostituzioni

Domanda

Due melanzane hanno esattamente lo stesso peso di tre peperoni.

Una melanzana e un peperone pesano insieme 300 grammi.

Sappiamo inoltre che tre melanzane pesano esattamente quanto una barbabietola e un peperone.

Dire quanti grammi pesano rispettivamente una melanzana, un peperone e una barbabietola.

Soluzione commentata

Siano:

M : peso di una melanzana

P : peso di un peperone

B : peso di una barbabietola

Impostiamo i dati ottenendo un sistema con 3 equazioni e 3 incognite:

$$\begin{cases} 2M = 3P \\ M + P = 300 \\ 3M = B + P \end{cases}$$

Risolvendo il sistema si ottiene la risposta

$$M = 180$$

$$P = 120$$

$$B = 420$$

Problema simile: 2006.3

Problema: LM 12.2011.2.F

Difficoltà: facile

Classificazione: problemi di calcolo

Parole chiave: equazioni di primo grado, sostituzioni, proporzioni

Domanda

Il valore in borsa di un'azione, della società Lisolachenonce Spa, ieri era aumentato del 20%, ma in questo momento sta perdendo il 20%.

Se vendo ora le 10 azioni, che avevo comperato l'altro ieri prima dell'aumento pagando in totale 150 euro, cosa mi succede ?

Risposte:

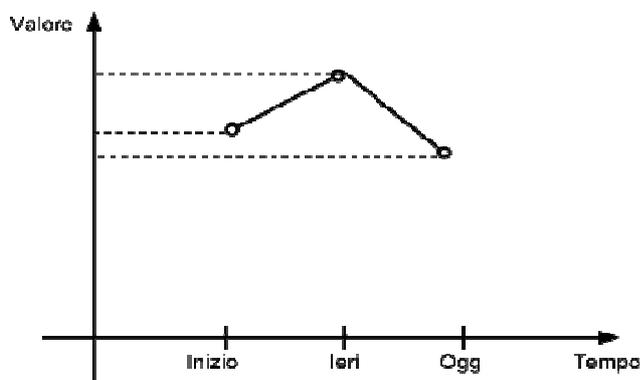
- a) Perdo 12 euro
- b) Perdo 6 euro
- c) Torno in pari
- d) Guadagno 6 euro

Soluzione commentata

Sia X il valore iniziale di un'azione, possiamo rappresentare quanto enunciato nel problema tramite la seguente tabella:

Tempo	Valore azione		Descrizione
Inizio (l'altro ieri)	$X = 150/20 = 7,5$ euro	Valore di una azione l'altro ieri	Situazione iniziale
Ieri	$Y = X + 0,2 X = 1,2 X$	Valore di una azione ieri	Aumento del 20% rispetto all'inizio
Oggi	$Z = Y - 0,2 Y = 0,8 Y = 0,96 X$	Valore di una azione oggi	Perdita del 20% rispetto a ieri

che graficamente equivale alla seguente situazione:



$$X = 7,5 \text{ euro}$$

$$Z = 0,96 * X = 0,96 * 7,5 = 7,2 \text{ euro}$$

Dai passaggi precedenti abbiamo ottenuto il valore iniziale di un'azione, 7,5 euro, ed il suo valore attuale, 7,2 euro.

Acquistando 20 azioni a 7,5 euro la spesa è di 150 euro.

Vendendo le 20 azioni a 7,2 euro recupero 144 euro.

Complessivamente quindi perdo 6 euro (risposta b).

Problema: LM 13.2011.3.F

Difficoltà: facile

Classificazione: problemi di calcolo

Parole chiave: equazioni di primo grado, sostituzioni

Domanda

Per spalare la neve caduta nel cortile della scuola, l'addetto Luigi impiega 6 ore, l'addetto Marco 10 ore e l'addetto Andrea 15 ore.

Quante ore impiegherebbero se si mettessero a spalare tutti insieme ?

Risposte:

- a) 1 ora e mezza
- b) 2 ore
- c) 3 ore
- d) 4 ore

Soluzione commentata

Soluzione intuitiva

Un approccio molto semplice, guardando alle possibili risposte riportate, è quello di considerare il tempo di Marco (10 ore) : se si fa aiutare da uno più lento (15 ore) e uno più veloce (6 ore) che mediamente hanno quasi la sua velocità il tempo originale da lui impiegato si ridurrà a circa un terzo: la risposta che più si avvicina è la c).

Soluzione grafica

Dalle risposte escludiamo la prima (1,5 ore) chiaramente non plausibile. Se la risposta deve essere un intero rappresentiamo graficamente l'area da spalare con 30 quadretti (che è il minimo comune multiplo tra 6,10 e 15) e indichiamo quanti di questi possono essere spalati da ogni addetto in un'ora.

L	L				
L	L				
L	M				
M	M				
A	A				

Luigi (L) in 6 ore spala 30 quadretti, quindi ne spala 5 in un'ora.
Marco (M) in 10 ore spala 30 quadretti, quindi ne spala 3 in un'ora.
Andrea (A) in 15 ore spala 30 quadretti, quindi ne spala 2 in un'ora.
Se spalassero insieme, in un'ora spalerebbero $5+3+2=10$ quadretti.
Per spalarne 30 avrebbero quindi bisogno di 3 ore (risposta c).

Soluzione formale

Possiamo rappresentare la domanda postaci attraverso la seguente equazione:

$$30 \text{ (quadretti)} = x + y + z$$

dove x , y e z sono rispettivamente i quadretti spalati da Luca, Marco e Andrea quando devono lavorare insieme per spalare tutta la superficie del cortile. Il numero 30 è stato scelto solo per avere dei numeri interi nei calcoli: potrebbe essere sostituito con qualsiasi numero e andrebbe bene lo stesso, a patto di modificare di conseguenza la velocità di spalatura.

L'area coperta da ogni addetto può essere calcolata attraverso il prodotto tra la "velocità" di spalatura, cioè quanti quadretti spala per unità di tempo, moltiplicata per il tempo di spalatura, cioè:

$$\text{Area_spalata} = \text{vel_spalatura} * \text{tempo}$$

Applicando quest'osservazione ai tre addetti otteniamo:

$$x = \text{vel}_L * T = 30/6 * T$$

$$y = \text{vel}_M * T = 30/10 * T$$

$$z = \text{vel}_A * T = 30/15 * T$$

siccome il tempo T è uguale per tutti poiché lavorano insieme possiamo sostituire nell'equazione iniziale ottenendo:

$$30 = 5T + 3T + 2T$$

che risolta nell'incognita T porta al risultato $T = 3$ ore (risposta c).

Problemi simili: 2005.1, 2005.2, 2008.5

Problema: LM 14.2009.2.M

Difficoltà: media

Classificazione: problemi di calcolo

Parole chiave: equazioni di primo grado, sistemi

Domanda

Ho cinque anni più di mia sorella, che ne ha 7 meno di nostra cugina. Quanti anni aveva nostra cugina quando la sua età era uguale alla somma delle nostre due?

Soluzione commentata

Siano:

x : età di mia sorella

y : la mia età

z : età di nostra cugina

Da notare che queste età sono le età all'istante T in cui si verifica la condizione che la cugina ha la somma delle età della protagonista e di sua sorella. Di fatto non ci interessa sapere quanto sia lontano T nel passato, ma i valori delle età in quell'istante.

Impostiamo un sistema di tre equazioni a tre incognite

$$\begin{cases} y = x + 5 \\ x = z - 7 \\ z = x + y \end{cases}$$

La prima equazione rappresenta che io ho cinque anni più di mia sorella, la seconda che mia sorella ha 7 anni meno di nostra cugina e la terza che l'età di nostra cugina è uguale alla somma della mia età e di quella di mia sorella

Risolvendo il sistema si ottiene

$$x = 2$$

$$y = 7$$

$$z = 9$$

Risposta: 9

Problemi simili: 2010.1, 2006.2

Problema: LM 15.2009.9.M

Difficoltà: media

Classificazione: problemi di calcolo

Parole chiave: calcolo mentale

Domanda

Se, nell'insieme dei numeri interi, vengono poste le seguenti condizioni:

$$a = b + 1$$

$$b = c - 2$$

$$c = d + 4$$

$$e = d + 2$$

Quale tra le seguenti affermazioni è vera?

Risposte:

a) $e > b$

b) $e = b$

c) $d > a$

d) $a < c < b$

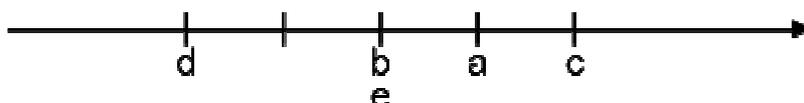
Soluzione commentata

Soluzione analitica

Supponiamo che sia vera la a): $e > b$
sostituiamo $d + 2 > c - 2$
per la 3) $d + 2 > d + 4 - 2$
da cui deriva che $d + 2 > d + 2$ evidentemente falsa
Da questa si deduce subito che $e = b$ (infatti siamo partiti da $e > b$, e siamo giunti ad una uguaglianza)
(risposta b)

Soluzione grafica

Un altro modo di procedere è quello di disegnare su una retta orientata le posizioni relative di **a**, **b**, **c**, **d** ed **e** e vedere quale delle affermazioni risulta soddisfatta. Per procedere si può disegnare il punto **a** e poi proseguire con gli altri, seguendo le condizioni imposte e ottenendo così il seguente disegno:



A questo punto è immediato vedere che la soluzione corretta è la b).

Problemi simili: 2006.7

Problema: LM 16.2011.1.M

Difficoltà: media

Classificazione: problemi di calcolo

Parole chiave: calcolo mentale, proprietà delle potenze

Domanda

Un'erba infestante si riproduce così rapidamente da raddoppiare ogni mese la superficie di terreno infestata. Una pianticella di questa erba, piantata in un campo, ha impiegato 12 mesi a infestarlo completamente. Quanti mesi sarebbero stati sufficienti se le pianticelle fossero state due ?

Soluzione commentata

Soluzione intuitiva 1

Molto semplicemente si potrebbe intuire che se dopo 12 mesi la pianta ha coperto una superficie raddoppiando ogni mese, all'undicesimo mese era la metà, quindi due piante avrebbero coperto due metà cioè il corrispondente di una pianta al dodicesimo mese (risposta 11).

Soluzione intuitiva 2

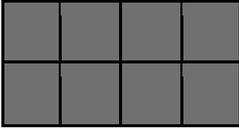
Partendo con una sola piantina al secondo mese se ne hanno due, che è proprio la condizione che ci viene chiesta nel problema, quindi basta solo togliere il primo mese alla soluzione precedente, ottenendo 11.

Soluzione formale

Vediamo ora invece un procedimento formale per giungere alla medesima soluzione.

Si supponga che un quadretto rappresenti la superficie occupata dalla pianticella inizialmente.

Mesi trascorsi	Superficie occupata	
1		2^1 quadretti
2		2^2 quadretti

3		2^3 quadretti
...
12		2^{12} quadretti

La superficie totale del campo quindi corrisponde a 2^{12} quadretti.

Se le piante fossero state 2, ogni mese i quadretti sarebbero stati 2×2^n (dove n è il numero dei mesi trascorsi)

Quindi per coprire la stessa superficie (2^{12}) avremmo avuto bisogno di n tale che:

$$\begin{aligned} 2 \times 2^n &= 2^{12} \\ 2^n &= 2^{12} / 2 \\ 2^n &= 2^{11} \\ n &= 11 \end{aligned}$$

Problema: LM 17.2011.4.M

Difficoltà: media

Classificazione: problemi di calcolo

Parole chiave: sistemi, media

Domanda

La media aritmetica dei quattro numeri 5, 9, X e Y vale 12.

Quanto vale la media aritmetica dei due numeri X+7 e Y-3 ?

Soluzione commentata

Ricordiamo che la media aritmetica di N numeri è data dalla somma dei numeri diviso N.

Sia M la media di X+7 e Y-3.

In questo caso quindi:

$$\begin{aligned} (1) \quad & (5+9+X+Y) / 4 = 12 \\ (2) \quad & (X+7+Y-3) / 2 = M \end{aligned}$$

Dalla (1), facendo qualche calcolo algebrico, si deduce che:

$$X + Y = 48 - 14 = 34$$

Dalla (2):

$$X + Y = 2M - 4$$

Quindi (risolvendo il sistema) deve essere:

$$\begin{aligned} 2M - 4 &= 34 \\ 2M &= 38 \end{aligned}$$

Da cui si ottiene $M = 19$.

Problema: LM 18.2009.6.M

Difficoltà: media

Classificazione: problemi di calcolo

Parole chiave: equazioni di secondo grado, sistemi, somma dei numeri da 1 ad N

Domanda

Un regista vuole sapere quante proiezioni del suo film sono state fatte in un certo cinema. L'usciera del cinema in cui il film è stato proiettato gli fornisce queste informazioni:

- alla prima proiezione c'era un solo spettatore;
- ad ogni nuova proiezione il numero degli spettatori è cresciuto di un'unità rispetto alla proiezione precedente.
- Il numero totale di spettatori durante tutte le proiezioni è stato 820

Soluzione commentata

Soluzione basata su calcoli

La relazione che lega il numero di proiezione con il numero di spettatori è rappresentata dalla tabella:

Numero proiezione	Numero spettatori
1	1
2	1 + 1
3	2 + 1
...
N	(N - 1) + 1 = N

Deve essere:

$$1+2+3+4+5+6+7+\dots+N = 820 \quad \text{dove } N \text{ è anche il numero della proiezione.}$$

Con carta e penna comincio a sommare i numeri naturali $1 + 2 + 3 + \dots$ fino ad ottenere 820. L'ultimo numero sommato (N) è 40.

Soluzione basata su approccio matematico

Se si conosce la formula che permette di calcolare la somma dei primi N numeri naturali, che è:

$$\sum_{1}^N n = \frac{N(N+1)}{2}$$

si può risolvere la seguente equazione:

$$820 = N(N+1)/2$$

che diventa:

$$N^2 + N - 1640 = 0$$

Applicando la nota formula per la soluzione delle equazioni di secondo grado si ottiene che la soluzione positiva (l'unica che abbia senso in questo problema) è appunto 40.

Problema: LM 19.2008.12.D

Difficoltà: media

Classificazione: problemi di calcolo

Parole chiave: calcolo mentale, insiemistica

Domanda

Incontro Aldo, che racconta: "In classe siamo in 20, ci hanno sottoposti a un test di Italiano e a uno di Matematica, ma solo in 6 li hanno superati entrambi. Io ho passato solo quello di Italiano, quelli che hanno superato solo Matematica sono stati il triplo di quelli che hanno passato solo Italiano".

Quanti, come minimo, sono stati insufficienti in entrambi gli esami?

Soluzione commentata

Dai dati risulta che 6 studenti hanno superato entrambi gli esami: quindi li possiamo escludere e così ne rimangono 14. Posto N il numero di studenti che hanno passato solo italiano, risulta che il numero di studenti che hanno passato solo matematica è 3N. Indicando con I il numero di studenti insufficienti in entrambe le materie, si ottiene la seguente equazione:

$$N + 3N + I = 14$$

da cui: $I = 14 - 4N$

Il problema si riduce a determinare il massimo valore di N tale che I rimanga positivo che si verifica essere $N=3$, dunque $I = 2$.

Problemi simili: 2002.9, 2005.4, 2005.7

Problema: LM20.2008.3.M

Difficoltà: media

Classificazione: problemi di insiemistica e calcolo combinatorio

Parole chiave: disposizioni con ripetizione, rotazioni

Domanda

Devo disegnare un quadrato, e per tracciare ciascun lato posso scegliere se usare un pennarello rosso oppure un pennarello blu.

Quanti quadrati diversi posso ottenere, tenendo presente che si considerano uguali i quadrati ottenuti da altro quadrato per rotazione?

Soluzione commentata

È necessario comprendere il significato dell'affermazione "si considerano uguali i quadrati ottenuti da altro quadrato per rotazione". Significa che la sequenza di colori non deve ripetersi; la sequenza può essere rappresentata dai lati L1,L2,L3,L4.

In questo caso, per esempio, la sequenza L3,L4,L1,L2 è ottenuta per rotazione.

Soluzione esaustiva

La soluzione si può ottenere dalla analisi delle $2^4 = 16$ disposizioni con ripetizione di 2 valori (ROSSO e BLU) su quattro posizioni (i quattro lati) :

L1	L2	L3	L4
ROSSO	ROSSO	ROSSO	ROSSO
ROSSO	ROSSO	ROSSO	BLU
ROSSO	ROSSO	BLU	ROSSO
ROSSO	ROSSO	BLU	BLU
ROSSO	BLU	ROSSO	ROSSO
ROSSO	BLU	ROSSO	BLU
ROSSO	BLU	BLU	ROSSO
ROSSO	BLU	BLU	BLU
BLU	ROSSO	ROSSO	ROSSO
BLU	ROSSO	ROSSO	BLU
BLU	ROSSO	BLU	ROSSO
BLU	ROSSO	BLU	BLU
BLU	BLU	ROSSO	ROSSO
BLU	BLU	ROSSO	BLU
BLU	BLU	BLU	ROSSO
BLU	BLU	BLU	BLU

Di questa tabella vanno conteggiate solo le righe che non sono rotazioni, ad esempio la seconda e la nona sono rotazioni dello stesso quadrato: partendo dall'alto e segnando su un foglio ogni nuovo quadrato incontrato, si arriva facilmente a determinare che sono solo 6 i quadrati diversi.

Soluzione intuitiva

Si può arrivare alla stessa soluzione ragionando sul problema:

- ci sono sicuramente due quadrati diversi, quelli composti da lati tutti blu o tutti rossi;
- ci sono altri due quadrati diversi, quelli composti da 3 lati di uno stesso colore e uno dell'altro colore.

Nessuno dei precedenti quadrati può essere ottenuto per rotazione dagli altri per il semplice fatto che hanno un numero di lati dei due colori che sono diversi tra loro.

Rimane solo la possibilità di avere due lati dello stesso colore e si hanno due alternative:

- un quadrato in cui i lati dello stesso colore condividono uno spigolo (RRBB ad esempio);
- un quadrato in cui i lati dello stesso colore non condividono nessuno spigolo (RBRB ad esempio)

Come si può vedere la soluzione è quindi $2+2+1+1 = 6$.

Problema simile: 2002.11

Problema: LM 21.2008.10.M

Difficoltà: media

Classificazione: problemi di insiemistica e calcolo combinatorio

Parole chiave: combinazioni, disposizioni con ripetizione

Domanda

Una paninoteca permette di creare il proprio panino scegliendo a piacimento tra 5 ingredienti: ogni ingrediente può essere incluso oppure no, indipendentemente dagli altri.

Ogni panino deve contenere almeno un ingrediente.

Quanti tipi diversi di panini si possono così formare?

Soluzione commentata

Soluzione esaustiva

Il problema si riduce a trovare quante diverse combinazioni semplici di n ingredienti su 5 si possono avere, con n che va da 1 a 5.

Per semplicità si può procedere isolando i singoli casi.

- I panini con un solo ingrediente sono ovviamente 5.
- Per i panini con due ingredienti presi dall'insieme dei 5 si possono contare quante possibili coppie diverse si formano in questo modo: ordinando gli ingredienti posso prendere il primo e con questo formare 4 coppie diverse; passo poi al secondo e con questo formo altre 3 coppie, sicuramente diverse dalle prime 4 poiché non contengono il primo elemento e procedendo in questo modo arrivo a formare altre 2 e un'ultima coppia. Riassumendo si avranno $4 + 3 + 2 + 1 = 10$ diversi panini con due ingredienti.
- Per i panini con tre ingredienti ho lo stesso numero che per quelli con due, poiché posso sempre pensare che a ogni panino con due ingredienti corrisponda un panino con 3 ingredienti "complementare", fatto cioè con gli ingredienti avanzati e quindi anche in questo caso si formeranno 10 panini diversi
- Per i panini con quattro ingredienti basta togliere di volta in volta un ingrediente e quindi si avranno 5 panini diversi.
- Infine c'è un solo tipo di panino con 5 ingredienti.

Sommando tutti i numeri ottenuti con i ragionamenti precedenti si ottiene $5 + 10 + 10 + 5 + 1 = 31$.

Soluzione che usa un approccio matematico

Dalla teoria degli insiemi si sa che l'insieme di tutti i sottoinsiemi di un insieme di 5 elementi è 2 alla quinta, cioè 32. Togliendo l'insieme vuoto si ottiene la risposta attesa, cioè 31.

Soluzione che usa un approccio informatico

Si può pensare di rappresentare la composizione degli ingredienti su una stringa di 5 bit. Se una cifra vale 1 significa che il corrispondente ingrediente c'è; se vale 0 che non c'è. Dato che i possibili numeri binari su 5 bit sono 32 la risposta sarà 31 perché bisogna escludere la stringa 00000.

Problema: LM 22.2010.2.M

Difficoltà: media

Classificazione: problemi di insiemistica e calcolo combinatorio

Parole chiave: combinazioni

Domanda

Se si lanciano due dadi, qual è la probabilità di fare 6 (naturalmente sommando i valori dei due dadi)?

Soluzione commentata

La probabilità che si verifichi un evento è il rapporto tra il numero dei casi favorevoli e quello dei casi possibili.

Supponiamo di avere due dadi regolari (D1 e D2) con 6 facce numerate da 1 a 6.

Nell'esercizio proposto i casi possibili sono 36 (la combinazione di ogni faccia di D1 con ogni faccia di D2), mentre i casi favorevoli sono 5, come si può vedere nello schema sottostante:

Casi possibili:

(1,1), (1,2), (1,3), (1,4), **(1,5)**, (1,6),
(2,1), (2,2), (2,3), **(2,4)**, (2,5), (2,6),
(3,1), (3,2), **(3,3)**, (3,4), (3,5), (3,6),
(4,1), **(4,2)**, (4,3), (4,4), (4,5), (4,6),
(5,1), (5,2), (5,3), (5,4), (5,5), (5,6),
(6,1), (6,2), (6,3), (6,4), (6,5), (6,6)

Casi favorevoli:

(1,5), (2,4), (3,3), (4,2), (5,1)

La probabilità di fare 6 lanciando due dadi è dunque $5/36$.

Problema: LM 23.2008.10.D

Difficoltà: difficile

Classificazione: problemi di insiemistica e calcolo combinatorio

Parole chiave: permutazioni

Domanda

Aldo, Bruno, Carlo e Dario vanno al cinema, dove occuperanno quattro posti consecutivi nella stessa fila.

In quanti modi possono sedersi, tenuto conto del fatto che Aldo e Bruno non si sopportano e pertanto non possono stare seduti vicini?

Risposte:

- a) 6
- b) 9
- c) 12
- d) nessuna delle precedenti

Soluzione commentata

Soluzione intuitiva

Le posizioni possibili avranno una delle seguenti 6 strutture in cui Aldo (A) e Bruno (B) non sono vicini:

AXYB
AXBY
XAYB
BXYA
BXAY
XBYA

dove X stanno per due posti vuoti che possono essere occupati indifferentemente dalla coppia (Carlo, Dario) o (Dario, Carlo). Quindi in totale si hanno 6×2 posizioni possibili. La risposta corretta è la c).

Soluzione esaustiva

Le posizioni possibili sarebbero 24 quelle in cui A e B sono vicini sono 12.

Posizioni possibili (permutazioni di 4 lettere senza ripetizione)

ABCD	ABDC	ACBD	ACDB	ADBC	ADCB
BACD	BADC	BCAD	BCDA	BDAC	BDCA
CABD	CADB	CBAD	CBDA	CDAB	CDBA
DABC	DACB	DBAC	DBCA	DCAB	DCBA

Togliendo quelle barrate, in cui A e B sono vicini ne rimangono 12. La risposta corretta è quindi la c).

Selezione di problemi di programmazione in C con soluzioni commentate

Premessa

Nelle edizioni delle Selezioni scolastiche delle Olimpiadi Italiane di Informatica del 2008, 2009, 2010 e 2011 sono stati assegnati complessivamente 34 problemi di programmazione. Di questi ne sono stati scelti 17 e ordinati per far seguire a un allievo curioso un percorso ispirato ai seguenti criteri:

- difficoltà crescente;
- primo apprendimento di concetti di programmazione o loro rinforzo se già noti;
- apprendimento del linguaggio C limitatamente alla parte che serve per superare le selezioni scolastiche.

Si consiglia di provare a risolvere da soli i problemi e di leggere solo dopo le soluzioni commentate.

Per alcuni problemi viene proposto di risolverne altri simili come utile esercizio.

Le soluzioni commentate dei problemi sono spesso precedute da alcuni richiami o puntualizzazioni, riconoscibili anche dall'uso di termini in *corsivo* per un chiarimento dei quali si rinvia alla *Guida al C per le Selezioni Scolastiche delle Olimpiadi di Informatica* allegata.

Tale guida è il naturale complemento a questo documento ed è utile soprattutto per coloro che intendono verificare la validità delle risposte tramite la reale esecuzione su calcolatore.

Ogni problema è caratterizzato da:

- a) un codice identificativo nella forma **C** *progressivo.anno.progressivoInterno.difficoltà*;
- b) un elenco di possibili temi in cui classificare il problema;
- c) un elenco di termini (parole chiave) del gergo di programmazione che vengono utilizzati significativamente nel commento alla soluzione.

Per qualche problema il livello di difficoltà è stato rivisto rispetto a quello originale attribuito nel testo della gara con il sistema dei punti.

Problema C1.2010.14.F

Difficoltà: facile

Classificazione: concetti di base, riconoscimento del problema risolto

Parole chiave: input/output, assegnamento, test, valutazione di espressioni

Si consideri il seguente frammento di programma C:

```
intr,c,s;

printf("Inserisci un numero intero compreso fra -10000 e 10000: ");
scanf("%d",&r);
c=1;
c=r*c;
s=1;
if (c<=r) {
    s=s+c;
    c=c*2;
}
printf("La variabile s vale %d\n",s);
```

Quale delle seguenti affermazioni è vera?

- a) Viene visualizzata la somma di tutti i numeri da 1 fino a $r + 1$
- b) Viene visualizzato il valore $r + 1$
- c) Viene visualizzato il valore $r + 1$ solo se $r \geq 1$
- d) Viene visualizzato il valore $2r + 1$

Soluzione commentata

Nelle selezioni scolastiche si propongono spesso solo dei frammenti di programmi invece di programmi completi per non distrarre l'attenzione dell'allievo con dettagli secondari che sono però necessari per l'effettiva esecuzione.

In domande come questa si chiede in sostanza di riconoscere il *problema* risolto dal *programma*, qui costituito da una breve *sequenza di istruzioni*. Occorre quindi eseguire mentalmente le istruzioni, una dopo l'altra, rappresentando e sintetizzando in qualche modo gli *effetti* prodotti da ciascuna delle singole istruzioni che compongono il frammento stesso.

L'istruzione `int r,c,s;` del C ha l'*effetto* di allocare spazio in memoria per tre *variabili*, di *tipo intero*, di nome `r,c,s` e il cui *valore* (o *contenuto*) iniziale è imprevedibile. Il programmatore può *inizializzare* il loro valore con delle istruzioni dirette o indirette di *assegnamento*.

Nel nostro caso l'istruzione `scanf("%d",&r)` del linguaggio C ha l'effetto di chiedere la *lettura da tastiera* di un valore intero e di assegnarlo alla *variabile intera* `r`. Il più delle volte avremo infatti a che fare solo con numeri interi, tra l'altro compresi tra un minimo negativo ed un massimo positivo.

L'istruzione `c=1;` ha invece l'effetto di assegnare il valore 1 alla variabile `c`. Il simbolo `=` si potrebbe leggere "diventa" e ha un significato diverso da quello di uguaglianza indicato in C con `==`.

L'istruzione `c=r*c;` ha l'effetto di assegnare il valore di `r` a `c` dato che moltiplicare 1 per qualsiasi numero `x` dà come risultato `x`.

L'istruzione `s=1;` ha naturalmente l'effetto di inizializzare la variabile `s` ad 1.

L'istruzione `if` esprime una *struttura di controllo condizionale*: si valuta una *condizione* e solo se è vera si esegue l'istruzione o il *blocco di istruzioni* seguente, cioè la sequenza compresa tra le parentesi graffe `{ }`.

Nel nostro caso la condizione `(c <= r)` è vera perché `c` contiene lo stesso valore della variabile `r`, cioè il numero immesso dalla tastiera.

La prima istruzione del blocco `s=s+c;` ha l'effetto di incrementare `s`, cioè di assegnare ad `s` il valore `r+1` dato che `s` contiene 1 e `c` contiene `r`. Il risultato è mostrato a video con l'istruzione:

```
printf("La variabile s vale %d\n",s)
```

in cui `%d` sta ad indicare che verrà inserita la rappresentazione in decimale di un numero specificato separatamente (nel nostro caso `s`). La coppia di caratteri `\n` serve per "andare a capo" riferito al video.

Si osservi che la seconda istruzione del blocco `c=2*c;` ha l'effetto di raddoppiare `c` ma è del tutto ininfluente ai fini della risposta alla domanda che è infatti la **b**).

In sostanza il frammento di programma è un modo volutamente complicato di risolvere il problema: "dato un intero trovare il successivo".

Un frammento più semplice che risolve lo stesso problema è il seguente:

```
int r;
printf("Inserisci un numero intero:");
scanf("%d",&r);
printf("Il successivo risulta %d\n",r+1);
```

Problema C2.2009.2.F

Difficoltà: facile

Classificazione: concetti di base

Parole chiave: assegnamento, ordine di valutazione delle espressioni

Si consideri il seguente frammento di programma:

```
main () {
    int a = 3;
    int b = 2;
    int c,d;
    c=2*a/b;
    d=2*(a/b);
    printf ("%d\n",c*d);
}
```

Quale tra i seguenti valori viene visualizzato a video dall'esecuzione di `main ()` ?

Risposte:

- a) 4
- b) 9
- c) 6
- d) 5

Soluzione commentata

Questo tipo di problemi hanno l'obiettivo di verificare se un programmatore è cosciente del fatto che ordini di valutazione diversi portano a risultati diversi. Mentre nell'algebra dei numeri reali espressioni come $x \cdot y / z$ esprimono lo stesso valore sia che si faccia prima la moltiplicazione o la divisione, nell'algebra dei *numeri interi* il risultato può essere diverso. In C e in molti linguaggi di programmazione si assume la regola che a parità di *priorità degli operatori* vale l'*ordine di valutazione* da sinistra a destra.

Nel nostro caso, nell'assegnamento $c=2*a/b$ viene prima fatto il prodotto $2*a$ e successivamente la divisione. Il risultato è quindi $6/2$ cioè 3. Nel secondo assegnamento $d=2*(a/b)$ le parentesi obbligano l'esecutore ad effettuare prima la divisione (a/b) cioè $3/2$ il cui risultato intero è 1 (e non 1.5 !). Quindi la variabile d contiene alla fine $2*1$ cioè 2.

Pertanto l'istruzione di scrittura a video del frammento produce $3*2 = 6$ (risposta c).

Si osservi che il frammento inizia con:

```
main(){
```

che sta ad indicare che le istruzioni successive costituiscono il corpo del *programma principale*, per distinguerlo dai *sottoprogrammi*, come vedremo nel prossimo problema.

Problemi simili: 2009.4 (programmazione), 2010.17

Problema C3.2008.3.F

Difficoltà: facile

Classificazione: concetti di base

Parole chiave: assegnamento, chiamata e definizione di funzione, passaggio di parametri, valutazione di espressioni

Dato il seguente frammento di codice:

```
int foo( int a, int b ) {
    return b - a;
}

main( ) {
    int a = 3, b = 5, x = 0;
    x = foo( b, a ) + a - b;
}
```

Quanto vale la variabile `x` alla fine dell'esecuzione?

Risposte:

- a) `x= 0`
- b) `x= 2`
- c) `x=-4`
- d) nessuna delle precedenti

Soluzione commentata

Il problema è facile per quanti hanno dimestichezza con il concetto di funzione matematica. Nel nostro caso viene prima definita (dichiarata) una *funzione*, un particolare tipo di *sottoprogramma*:

```
int foo( int a, int b ) {
    return b - a;
}
```

La prima riga descrive l'*interfaccia della funzione*, cioè le informazioni necessarie per poterla utilizzare. Nel nostro caso ci dice che:

- il suo calcolo deve fornire (*restituire*) un numero intero;
- la funzione si chiama `foo`;
- `a` e `b` sono *parametri* interi, assunti in C come *variabili locali* al sottoprogramma inizializzate con i dati da elaborare, specificati al momento della *chiamata*.

Dentro le graffe `{ }` si mette il *corpo della funzione* che nel nostro caso è costituito semplicemente da:

```
{
    return b - a;
}
```

`return espressione` in una funzione C ha due effetti: conclude l'elaborazione e restituisce all'utilizzatore il valore di *espressione* come risultato dell'elaborazione. Nel nostro caso restituisce la differenza tra il secondo e il primo *parametro*.

Ad esempio, il valore restituito da `foo(30,70)` sarebbe 40. Nel nostro caso la funzione `foo` viene *richiamata* dentro il `main` nell'espressione che sta a destra del simbolo di assegnamento:

```
x = foo( b, a ) + a - b;
```

Dato che le variabili `a` e `b` del `main` (da non confondere con i parametri `a` e `b` di `foo`, incidentalmente con lo stesso nome) sono inizializzate rispettivamente a 3 e 5, l'assegnamento sopra equivale a:

```
x = foo(5,3) + 3 - 5;
```

e quindi, *sostituendo* `foo(5,3)` con il suo valore, si desume che `x` conterrà `- 2 + 3 - 5`, cioè -4.

La risposta esatta è quindi la c). In problemi successivi vedremo che le funzioni possono svolgere elaborazioni più complesse (con `if`, cicli, ricorsione, ecc.).

Si noti che ogni parametro assume il valore corrispondente **per posizione**: al momento della chiamata (il *primo* parametro assume il *primo* valore, il *secondo* parametro assume il *secondo* valore, ecc.).

Problema C4.2011.6.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: assegnamento, post-incremento, ciclo while, operazione modulo

È dato il seguente programma:

```
#include <stdio.h>
int i,c;
main() {
    c=0; i=0;
    while (i<100) {
        c++;
        if (c % 2 != 0) i++;
    }
    printf("c=%d\n",c);
}
```

Cosa viene visualizzato a video dall'esecuzione di main() ?

Risposte:

- a) c=100
- b) c=101
- c) c=199
- d) il ciclo while non termina mai, quindi non viene visualizzato nulla

Soluzione commentata

Questo problema introduce l'istruzione *while*, presente con lo stesso significato ma con piccole varianti sintattiche in tutti i linguaggi di programmazione. L'istruzione *while* esprime una *struttura di controllo iterativa*: comporta infatti la ripetizione ciclica di un'istruzione o di un gruppo di istruzioni detto *corpo del ciclo*. Prima di ogni esecuzione del corpo viene verificata una *condizione* detta *guardia del while*: se è falsa l'intera istruzione *while* ha termine.

Nel nostro caso la guardia è costituita dalla condizione: ($i < 100$).

Il corpo del ciclo è costituito dal blocco di due istruzioni:

```
{
    c++;
    if (c % 2 != 0) i++;
}
```

La prima istruzione (detta *post incremento*) ha l'effetto di incrementare la variabile *c* (come in: $c=c+1$;). La seconda ha l'effetto di incrementare la variabile *i* solo se *c* è dispari. La condizione ($c \% 2 \neq 0$) è infatti da leggersi come "il resto della divisione di *c* per 2 è diverso da 0". L'operazione resto è meglio nota in informatica come *modulo* tra numeri.

In pratica *i* viene incrementata un ciclo sì e uno no fino a quando la condizione $i < 100$ risulta falsa, vale a dire fino a quando la variabile *i* assume il valore 100, visto che *i* parte da 0.

Sono quindi necessarie 200 *iterazioni* del ciclo *while* prima che *i* raggiunga la condizione per uscire. Visto che *c* parte da 0, il valore finale di *c* al 200-esimo ciclo sarà 199.

Alternativamente si osservi che dopo ogni due iterazioni la relazione matematica esistente tra *c* ed *i* è

$c = 2i - 1$. Quindi, all'uscita del *while*, quando *i* vale 100, *c* vale 199.

La risposta esatta è quindi la c).

Problema C5.2008.2.F

Difficoltà: facile

Classificazione: concetti di base

Parole chiave: assegnamento, ciclo con almeno un'iterazione

Data la seguente porzione di codice:

```
int sum = 0, contatore = 1000;
do
{
    sum = 0;
    sum += --contatore;
} while( contatore < 0 );
```

Quanto vale la variabile `sum` alla fine dell'elaborazione ?

Risposte:

- a) 1
- b) 999
- c) 1000
- d) 1001

Soluzione commentata

Il corpo di un'istruzione `while` può non essere eseguito neanche una volta se la guardia è subito falsa. Se invece si vuole che venga eseguito almeno una volta conviene prevedere il controllo della condizione subito dopo l'esecuzione del corpo. A tale proposito il C dispone della struttura di controllo *do-while*, usata in questo problema.

Il corpo del ciclo *do-while* è racchiuso tra le parole chiave `do` e `while` e nel nostro caso è composto da due istruzioni:

```
{
    sum = 0;
    sum += --contatore;
}
```

La seconda è un'istruzione con una *forma compatta tipica del C (idioma C)* equivalente alle due istruzioni :

```
contatore = contatore - 1;
sum = sum + contatore;
```

Si osservi che, considerando anche l'assegnamento `sum = 0`, il corpo del ciclo è quindi equivalente a:

```
{
    contatore = contatore-1;
    sum = contatore;
}
```

Nel nostro caso `sum` assume nel primo ciclo il valore 999. Il test `(contatore < 0)` del `do-while` è falso dato che `contatore` contiene 999 e quindi non si rientra in ciclo. Pertanto la risposta esatta è la b).

Problema C6.2010.15.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: ciclo `while`, ciclo con almeno un'iterazione

Si consideri il seguente frammento di programma:

```

main() {
    int n,i,j,a=0,b=0;
    printf("Inserisci un numero intero: ");
    scanf("%d",&n);
    if (n<0) n=-n;
    i=j=n;
    while (i>0) {
        a+=1; i--;
    }
    do {
        b+=1; j--;
    }
    while (j>0);
    printf ("ca=%d b=%d\n",a,b);
}

```

Cosa viene visualizzato a video dall'esecuzione di main() se alla domanda Inserisci un numero intero: viene inserito da tastiera il valore 100 ?

Cosa viene visualizzato a video dall'esecuzione di main() se alla domanda Inserisci un numero intero: viene inserito da tastiera il valore 0 ?

Soluzione commentata

Questo tipo di problemi si propone di verificare le conoscenze relative alla differenza tra le due *strutture di controllo iterative* while e do-while del C. Dopo aver letto un numero intero nelle variabile n e averne fatto il valore assoluto (istruzione if (n<0) n=-n;) le variabili i e j vengono inizializzate ad n con un *assegnamento multiplo*.

Il ciclo while conta in a il numero di cicli necessari per rendere falsa la sua guardia in ingresso (i>0) decrementando ogni volta i.

Il successivo ciclo do-while conta in b il numero di cicli necessari per rendere falsa la sua guardia in uscita (j>0) decrementando ogni volta j.

Se n è maggiore di 0 i valori di i e di j coincideranno. Ma se n vale 0 dal while si esce subito mentre dal do-while si esce solo dopo aver eseguito il corpo.

Pertanto la risposta corretta è:

a=100 b=100

a=0 b=1

Problema C7.2008.7.M

Difficoltà: media

Classificazione: concetti di base, riconoscimento problema risolto

Parole chiave: funzione, ciclo for, tavola di traccia

Si consideri la seguente funzione:

```

int foo( int N ) {
    int i, R = 1;
    for ( i = 0; i < N; i++ )
        R = R + R;
    return R;
}

```

Indicare qual è il valore restituito dall'invocazione di foo(6).

Risposte:

a) 32

b) 64

c) 128

d) nessuno dei precedenti

Soluzione commentata

Il *for* è un'altra istruzione per esprimere cicli usata tipicamente quando si vuole ripetere il corpo in modo che ad ogni ciclo una variabile intera (detta *indice del for*) assuma ordinatamente dei valori compresi in un certo intervallo. Nel nostro caso l'intervallo è compreso tra 0 ed $N-1$.

Un *for* si può sempre sostituire con il *while*. Nel nostro caso il *while* equivalente sarebbe:

```
i=0;  
while (i < N)  
{  
    R = R + R;  
    i++;  
}
```

in cui le parti sottolineate sono quelle espresse più concisamente nel *for*.

Osserviamo che $R = R + R;$ può essere espressa come $R = 2 * R;$. Tale istruzione viene eseguita N volte e visto che R contiene inizialmente 1 l'effetto finale sarà quello di restituire 2^N . Possiamo rendercene meglio conto costruendo la cosiddetta *tavola di traccia* assegnando ad N il valore 6:

Istruzione/ condizione	Esito	i	R	N
				6
$R = 1$			1	
$i = 0$		0		
$i < N$	vero			
$R = R + R$			2	
$i++$		1		
$i < N$	vero			
$R = R + R$			4	
$i++$		2		
$i < N$	vero			
$R = R + R$			8	
$i++$		3		
$i < N$	vero			
$R = R + R$			16	
$i++$		4		
$i < N$	vero			
$R = R + R$			32	
$i++$		5		
$i < N$	vero			
$R = R + R$			64	
$i++$		6		
$i < N$	falso			

La risposta corretta è quindi la b).

Problemi simili:2008.5 (programmazione)

Problema C8.2008.1.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: procedura, passaggio per riferimento, indirizzo di memoria di una variabile

Si consideri il seguente frammento di codice:

```

void foo( int *a, int b) {
    int temp = *a;
    *a = b;
    b = temp;
};

main( ) {
    int a = 1;
    int b = 5;
    foo( &a, b );
}

```

Quanto valgono le variabili `a` e `b` alla fine dell'esecuzione?

Soluzione commentata

Alcuni linguaggi distinguono tra due tipi di sottoprogrammi: le funzioni e le *procedure*. I primi, come abbiamo già visto (problema C2.2009.2.F), restituiscono un valore.

Le procedure invece non restituiscono alcun valore e sono riconoscibili in C per iniziare con la parola chiave **void**. Le procedure (chiamate comunque impropriamente "funzioni" nel gergo del C) producono degli effetti sull'*ambiente di calcolo*: tipicamente sul video o sulle variabili contenute nella memoria del computer.

Una semplice procedura che modifica il video è quella che scrive un messaggio a video:

```

void saluta() {
    printf("Salve \n");
}

```

Mentre una funzione viene chiamata all'interno di un'espressione (nella quale il valore restituito dalla funzione viene usato per valutare l'espressione stessa) una procedura viene chiamata come se fosse una nuova istruzione (ad es. `saluta();`)

Questo tipo di problema si propone di verificare le conoscenze relative alla differenza tra *passaggio per valore* e *passaggio per riferimento* (o *indirizzo*) nei sottoprogrammi.

In molti linguaggi di programmazione i parametri costituiti da una variabile possono essere passati ad una procedura (o ad una funzione) in due modi:

- per *valore* : viene passata la copia del valore della variabile del programma chiamante: la variabile del programma chiamante e la variabile della procedura chiamata sono due variabili indipendenti, e occupano locazioni di memoria diverse;
- per *riferimento*¹: viene passato in pratica l'*indirizzo* della locazione di memoria che contiene il valore della variabile del programma chiamante; la variabile del programma chiamante e la variabile della procedura coincidono perché si riferiscono alla stessa locazione di memoria.

Nel nostro problema vogliamo che gli effetti della procedura `foo` vengano prodotti sulla variabile `a` del `main`. Per raggiungere questo scopo è necessario che `foo` "conosca" l'*indirizzo di memoria* di `a`, esprimibile con l'espressione `&a`. Il C prevede che tra i tipi di dato di un parametro possa esserci il tipo "indirizzo di una variabile intera" con la notazione `int *`. Per accedere al contenuto del parametro nel corpo della procedura è necessario anteporre il simbolo `*` al nome del parametro (operazione di *dereference*: dato l'indirizzo reperire il contenuto). Quindi la prima istruzione del corpo di `foo`:

```
int temp = *a;
```

ha l'effetto di caricare in `temp` il valore della variabile `a` del `main`, cioè 1.

L'istruzione successiva:

```
*a = b;
```

ha l'effetto di caricare nella variabile `a` del `main` il valore del parametro `b`, assimilabile ad una variabile locale di `foo`, cioè 5 che è il valore, passato dal `main`, contenuto nella omonima (accidentalmente) variabile `b` del `main`. Infine l'istruzione:

¹ In C non esiste un passaggio di riferimento vero e proprio, ma si usa il meccanismo del passaggio di indirizzo per ottenere lo stesso risultato.

```
b = temp;
```

carica in `b` di `f00` il valore originale di `a`, cioè 1. Tale istruzione è del tutto inutile perché non produce effetti sull'ambiente di calcolo una volta conclusa l'esecuzione del corpo della procedura.

In definitiva, l'effetto prodotto all'esterno da `f00` è quello di assegnare ad `a` del `main` il valore di `b` del `main`.

La risposta è quindi: `a=5` e `b=5`.

Si osservi che si sarebbe potuto semplificare il corpo di `f00` semplicemente con:

```
*a = b;
```

Problemi simili: 2009.3 (programmazione)

Problema C9.2008.9.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: ricorsione, if-then-else

Si consideri la seguente funzione:

```
int ric( int x ){
    if (x == 1)
        return x;
    else
        return ric( x-1 ) + 2*x - 1;
}
```

Dire cosa restituisce l'invocazione di `ric(10)`

Risposte:

a) 28

b) 99

c) 101

d) nessuna delle precedenti

Soluzione commentata

La funzione `ric` riportata nell'esempio è *ricorsiva*, cioè sfrutta il concetto di *ricorsione*: significa che al suo interno contiene una chiamata a se stessa. Una funzione ricorsiva correttamente concepita prevede che ci siano sempre dei *cas* base in cui il risultato viene prodotto direttamente, senza effettuare la chiamata ricorsiva.

Noi abbiamo un unico caso base per `x==1`: si restituisce lo stesso `x`, cioè 1.

Altrimenti (ramo *else* dell'*if*) in tutti gli altri casi (*cas non base*), la funzione richiama se stessa con parametro `x-1`; il valore ottenuto viene poi sommato a `2*x-1` prima di essere restituito al chiamante.

Facciamo un piccolo esempio calcolando `ric(3)` invece di `ric(10)`.

$ric(3) \rightarrow ric(2) + 2*3 - 1 \rightarrow (ric(1) + 2*2 - 1) + 3*2 - 1 \rightarrow 1 + 3 + 5 \rightarrow 9$

Ogni caso non base cerca di riportarsi ad un caso non base più semplice fino a raggiungere il caso base cioè `ric(1)`.

Si può allora indurre che `ric(10)` calcoli la somma dei primi 10 numeri dispari: `1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + 17 + 19`. Tale somma vale 100 e quindi la risposta esatta è la d).

Ci si poteva arrivare anche più rapidamente sapendo che la somma dei primi `N` numeri dispari è uguale al quadrato di `N`.

Problema C10.2009.7.D

Difficoltà: difficile

Classificazione: ricorsione

Parole chiave: procedura, conversione in binario

È dato il seguente frammento di programma:

```
void ric (int n) {
    int m;
    m = n/2;
    if (m != 0) ric (m);
    printf ("%ld",n % 2);
}

main( ) {
    ric (729);
}
```

Cosa viene visualizzato a video dall'esecuzione di main ()?

Soluzione commentata

Anche le procedure possono essere ricorsive, come quella proposta in questo problema. I casi base si hanno quando n è 0 oppure 1, ossia quando $n/2$ è uguale a 0 (divisione intera, ricordiamolo) rendendo così falsa la condizione dell'if e non eseguendo la chiamata ricorsiva. Nei casi base il programma scrive a video n stesso (in quanto $n\%2$ coincide con n).

Per i casi non base ($n>1$) viene effettuata la chiamata ricorsiva con parametro $n/2$ e scritto poi il resto della divisione di n per 2.

Chi ha fatto delle conversioni manuali da decimale a binario potrebbe avere il sospetto che si tratti della versione ricorsiva della procedura di conversione. Proviamo per un numero più basso di quello dato organizzando una traccia per le procedure ricorsive:

```
inizioric(6)
    m=n/2;           m contiene 3
    inizioric(3);
        m=n/2;       il nuovo m contiene 1
        inizioric(1); (caso base)
            print (n % 2); scrive 1
        fineric(1)
            print(n %2);   scrive 1 perché n ora vale 3
    fineric(3)
        print(n % 2);     scrive 0 perché n ora vale 6
fineric(6)
```

Ad ogni chiamata ricorsiva si incolonna la traccia un po' più a destra per tener conto che si entra in un ambiente locale di calcolo nuovo e più recente: i "vecchi" n ed m rimangono "congelati" fino al completamento della chiamata ricorsiva interna.

L'output a video è dunque 110 che è effettivamente la conversione binaria di 6.

Vengono infatti riprodotti in ordine inverso i resti delle successive divisioni per 2 del numero dato in ingresso. A questo punto applichiamo decisamente al numero 729 la nota tecnica manuale delle conversioni in binario:

```
729 | 1
364 | 0
182 | 0
 91 | 1
 45 | 1
 22 | 0
 11 | 1
  5 | 1
  2 | 0
  1 | 1
  0
```

e otteniamo la risposta richiesta 1011011001 mediante una lettura dal basso all'alto dei resti.

Problema C11.2010.19.D

Difficoltà: difficile

Classificazione: ricorsione

Parole chiave: funzioni

Si consideri il seguente frammento di programma:

```
int succ(int i) {
    if (i<=2)
        return(1);
    else
        return(3*succ(i-1)+2*succ(i-2)-succ(i-3));
}

main() {
    printf("num=%d\n",succ(7));
}
```

Cosa viene visualizzato a video dall'esecuzione di main()?

Soluzione commentata

Si riconosce una funzione ricorsiva in cui i casi base si hanno per $i \leq 2$, per i quali si restituisce il valore 1. I casi non base richiedono 3 chiamate ricorsive con parametri $i-1$, $i-2$ e $i-3$.

Quindi $\text{succ}(7)$ sarà dato da: $3 * \text{succ}(6) + 2 * \text{succ}(5) - \text{succ}(4)$. Osserviamo che $\text{succ}(6)$ sarà dato da: $3 * \text{succ}(5) + 2 * \text{succ}(4) - \text{succ}(3)$. Per evitare di calcolare più volte lo stesso $\text{succ}(x)$ conviene organizzare il calcolo partendo "dal basso" (cioè da $i=1$) riempiendo progressivamente una tabella a due colonne:

i	succ(i)
0	1 (caso base)
1	1 (caso base)
2	1 (caso base)
3	$3*1+2*1-1 = 4$
4	$3*4+2*1-1 = 13$
5	$3*13+2*4-1 = 46$
6	$3*46+2*13-4 = 160$
7	$3*160+2*46-13 = 559$

In questo modo quando si ha bisogno dei tre succ "precedenti" basta andarli a prendere nelle 3 righe precedenti.

In definitiva 559 è la risposta da dare.

Problema C12.2010.18.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: array, vettore, divisione e modulo tra interi

Si consideri il seguente frammento di programma:

```

#define D 10
main() {
    int M[D],i;
    for (i=0;i<D;i++)
        M[i]=9*i;
    printf("Sequenza=");
    for (i=0;i<D;i++)
        if (M[i]%(i+1)==0)
            printf(" %2d",M[i]/(i+1));
        else
            printf(" %2d",M[i]%(i+1));
}

```

Cosa viene visualizzato a video dall'esecuzione di main()?

Soluzione commentata

In questo frammento di programma appare un *array* di 10 interi nella riga:

```
int M[D],i;
```

dove *D* sta per 10 come dichiarato in `#define D 10`.

L'array *M* è un gruppo di variabili, chiamate *elementi*, che hanno un nome in comune (*M*, nel nostro caso) e un nome individuale costituito da un *indice* intero compreso, nel nostro caso, tra 0 e 9. Per individuare un elemento del gruppo si fa uso della notazione *M[indice]*. Così *M[0]* rappresenta il primo elemento del gruppo ed *M[9]* l'ultimo elemento del gruppo.

Gli array sono noti anche come *vettori* quando, come in questo caso, si fa uso di un solo indice per identificare un elemento del gruppo. La "potenza" degli array è rappresentata dal fatto che l'indice può essere un'espressione. In particolare può essere una variabile chiamata *variabile indice*, come avviene in:

```
for (i=0;i<D;i++)
    M[i]=9*i;
```

che ha l'effetto di inizializzare i 10 elementi di *M* con i numeri 0,9,18,27,36,45,54,63,72,81 attraverso l'uso della variabile indice *i*. Il `for` successivo elabora uno ad uno i suddetti elementi chiedendosi nel corpo del ciclo:

"*M[i]* è divisibile per (*i+1*)?" Se sì scrivi il *quoziente* altrimenti scrivi il *resto* della divisione.

Convienne allora organizzare la traccia del calcolo in una tabella come la seguente:

i	M[i]	i+1	M[i]%(i+1)	M[i]/(i+1)	output
0	0	1	0	0	0
1	9	2	1		1
2	18	3	0	6	6
3	27	4	3		3
4	36	5	1		1
5	45	6	3		3
6	54	7	5		5
7	63	8	7		7
8	72	9	0	8	8
9	81	10	1		1

Quindi l'output atteso a video è:

Sequenza= 0 1 6 3 1 3 5 7 8 1.

Problema C13.2011.10.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: array

Si consideri il seguente frammento programma:

```
#define N 9

int R[N];
int i=0, j=(N-1), c=0, k;
while(i<j)
  if(R[i]+R[j] > k){
    c += (j-i);
    i++;
  }
  else j--;
```

Dire cosa contiene la variabile *c* al termine dell'esecuzione del frammento di programma, quando:

```
R = [100,87,67,54,34,23,11,10,1];
k = 143;
```

Soluzione commentata

I due indici *i* e *j* vengono inizializzati per indicare rispettivamente il primo e l'ultimo elemento di *R*. Nel while gli indici si possono muovere da sinistra verso destra (*i++*) e da destra verso sinistra (*j--*) fino ad "incontrarsi". Nel corpo del while si verifica se la somma tra *R[i]* ed *R[j]* è superiore ad una certa soglia *k* (nel nostro caso 143): in caso affermativo si accumula in *c* la differenza (*j - i*) a quelle precedentemente ottenute e si sposta *i*, altrimenti si sposta *j*.

La prima condizione positiva è:

[100,87,67,54,34,23,11,10,1] quindi con *i*=0 e *j*=3, *c* contiene 3

La seconda posizione positiva è:

[100,87,67,54,34,23,11,10,1] quindi con *i*=1 e *j*=2, *c* contiene 3 + (2-1)

La risposta da dare è quindi: *c*= 4

Problemi simili: 2010.20 (programmazione)

Problema C14.2011.9.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: array, accesso indiretto

Si consideri il seguente frammento di programma:

```
main(){
  int T[25], S[10], W[5], i;
  i=0;
  while(i<=24) S[T[i++]-1]++;
  i=0;
  while (i<= 9) W[S[i++]-1]++;
}
```

Si supponga che i vettori *T*, *S*, e *W* inizialmente contengano i seguenti valori:

```
T = [1,1,2,2,2,3,3,3,4,4,4,5,6,6,7,7,8,8,8,8,8,9,10,10,10];
S = [0,0,0,0,0,0,0,0,0,0];
W = [0,0,0,0,0];
```

Scrivere il contenuto del vettore *W* dopo l'esecuzione del programma.

Soluzione commentata

Questo è un esempio di frammento di programma che contiene *accessi indiretti* ad un array attraverso il contenuto di un altro array come in $A[B[x]]$.

L'accesso è costruito attraverso i seguenti passi: $x \rightarrow B[x] \rightarrow A[B[x]]$.

Il primo ciclo potrebbe essere espresso meno concisamente ma più chiaramente con:

```
i=0;
while(i<=24)
{
    int k=T[i]-1; /*con i++ l'incremento di i avviene dopo la valutazione */
    i=i+1;
    S[k]=S[k]+1;
}
```

L'array S è un array di contatori inizialmente tutti azzerati. Per ogni elemento di T viene calcolata la posizione k di S a cui corrisponde il contatore da incrementare.

In questo modo all'uscita del while avremo:

$S = [2,3,3,3,1,2,2,5,1,3]$

Quindi in $S[i]$ compare il numero di volte che il valore $(i+1)$ compare in T .

Nel secondo while viene applicato lo stesso procedimento tra S e w : in $w[i]$ ci sarà il numero di volte che $(i+1)$ compare in S . Quindi in definitiva la risposta è:

$w = [2,3,4,0,1]$

Problema C15.2008.4.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: matrici, righe, colonne, passaggio di matrici, visita, cicli, for annidati

Si consideri il seguente frammento di codice:

```
void esegui( int M[][5], int C, int R1, int R2 ) {
    int i;
    for ( i = 0; i < C; i++ )
        M[R1][i] = M[R1][i] + M[R2][i];
}

main() {
    int i, M[5][5];
    for ( i = 0; i < 2; i++ )
        esegui( M, 5, i, 3 );
}
```

La matrice M inizialmente contiene tutti 1 nella prima riga, tutti 2 nella seconda riga e così via.

Indicare il contenuto della matrice M al termine del programma.

Risposte:

a) 5 5 5 5 6 6 6 6 3 3 3 3 4 4 4 4 5 5 5 5	b) 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5	c) 5 5 5 5 4 4 4 4 3 3 3 3 2 2 2 2 1 1 1 1	d) 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5
--	--	--	--

Soluzione commentata

Una *matrice* è un array in cui si identifica ciascun elemento mediante due indici invece che con uno solo come nei *vettori* considerati negli array presenti nei problemi precedenti.

Il primo indice viene convenzionalmente chiamato *indice di riga* e il secondo *indice di colonna* immaginando la matrice come costituita da righe e colonne.

Nel nostro caso la matrice ha 5 righe e 5 colonne con entrambi gli indici che vanno da 0 a 4.

Dal testo si assume che lo stato iniziale della matrice sia:

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Questo obiettivo, non richiesto dal testo, è ottenibile con due cicli *for annidati* (cioè uno dentro all'altro):

```
int r,c;
for (r=0;r<5;r++)
  for(c=0;c<5;c++)
    M[r][c]= r + 1;
```

secondo un classico schema di *visita della matrice per righe*.

La procedura *esegui* richiede una matrice come primo parametro. Il linguaggio C, per come accede agli elementi di una matrice, richiede che venga specificato il numero di colonne (nel nostro caso 5) :

```
void esegui( int M[][5], int C, int R1, int R2 )
```

Il corpo della procedura:

```
for ( i = 0; i < C; i++ )
  M[R1][i] = M[R1][i] + M[R2][i];
```

ha l'effetto di incrementare i primi C elementi della riga di indice R1 con il corrispondente elemento (cioè nella stessa colonna) della riga R2.

Quindi:

```
esegui( M, 5, i , 3 );
```

con $i = 0$ avrà l'effetto di modificare la matrice M come segue:

```
5 5 5 5 5
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

sommando cioè la riga di indice 3 a quella di indice 0.

La successiva e ultima chiamata :

```
esegui( M, 5, i , 3 );
```

con $i = 1$ avrà l'effetto di sommare sempre la riga di indice 3 a quella di indice 1:

```
5 5 5 5 5
6 6 6 6 6
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Quindi la risposta esatta è la a).

Problema C16.2011.8.M

Difficoltà: media

Classificazione: concetti di base

Parole chiave: matrici, ordini di visita

È dato il seguente programma:

```
#include <stdio.h>
#define nrig 5
#define ncol 5
int i,j,m[nrig][ncol];

main() {
    for (i=0;i<nrig;i++)
        for (j=0;j<ncol;j++)
            m[i][j]=i*j+j;
    for (j=ncol-1;j>=0;j--) {
        for (i=nrig-1;i>=0;i--)
            printf("%d ",m[i][j]);
        printf("\n");
    }
}
```

Cosa viene visualizzato a video dall'esecuzione di main() ?

Risposte:

a) 20 16 12 8 4 15 12 9 6 3 10 8 6 4 2 5 4 3 2 1 0 0 0 0 0	b) 0 0 0 0 0 1 2 3 4 5 2 4 6 8 10 3 6 9 12 15 4 8 12 16 20	c) 4 8 12 16 20 3 6 9 12 15 2 4 6 8 10 1 2 3 4 5 0 0 0 0 0	d) 0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4
---	---	---	---

Il primo for ha l'effetto di inizializzare la matrice m con una visita per righe con il valore di $i*j+j$:

```
0 1 2 3 4
0 2 4 6 8
.....
.....
0 5 10 15 20
```

Il secondo for fornisce come primo output il contenuto di $m[4][4]$ cioè 20. L'unica tra le 4 risposte che ha 20 come primo output è la a).

Problema C17.2011.12.D

Difficoltà: difficile

Classificazione: ricorsione

Parole chiave: procedura, albero delle chiamate

È dato il seguente programma:

```

#include <stdio.h>

void f(int T[], inti, int dim) {
    if (i<dim) {
        f(T,2*i+1,dim);
        printf ("%d",T[i]);
        f(T,2*i+2,dim);
    }
}

main() {
    int Q[8]={1,2,3,4,5,6,7,8};
    printf ("ris = ");
    f(Q,0,8);
}

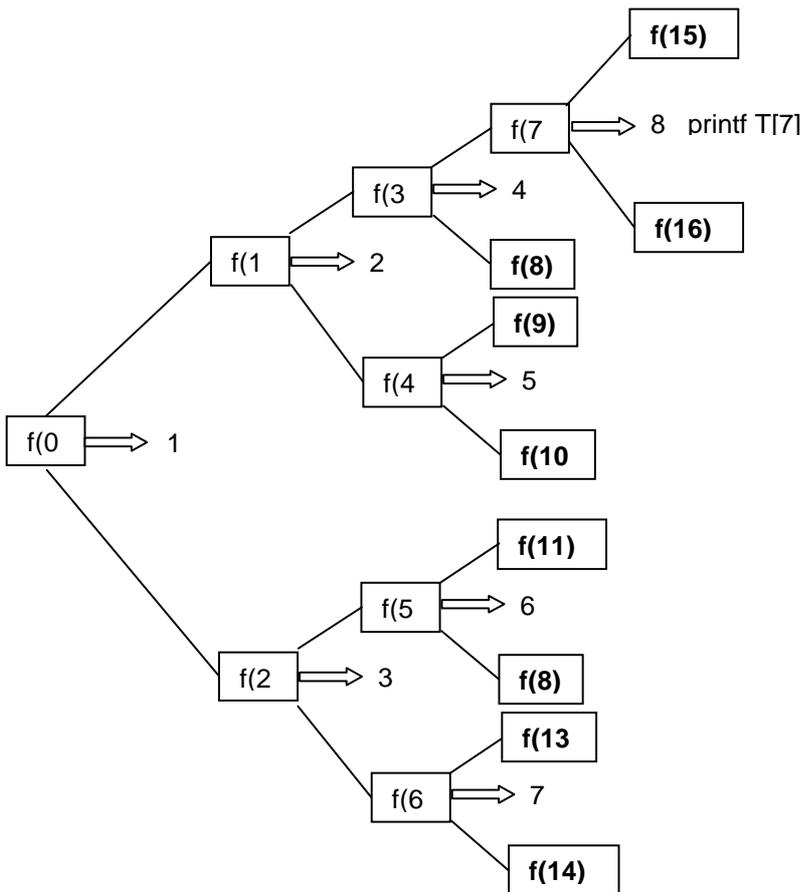
```

Cosa viene visualizzato a video dall'esecuzione di main()?

Soluzione commentata

Si tratta di eseguire una procedura ricorsiva. Il problema è complicato dal fatto che le chiamate ricorsive sono 2 e l'output è in mezzo alle due. I casi base si hanno con $i \geq dim$ per i quali non viene prodotto alcun effetto.

Un modo abbastanza semplice e veloce di venirne fuori senza "perdersi" (ci sono 17 chiamate) è quello di costruire l' *albero delle chiamate* e riprodurre gli output (evidenziati con una doppia freccia) dall'alto al basso. Nella figura i casi base sono stati evidenziati in grassetto.



La risposta è quindi:

ris = 8 4 2 5 1 6 3 7.

Guida al C per le selezioni scolastiche delle Olimpiadi di Informatica

Premessa

Lo scopo di questa breve guida (o “tutorial”) è di permettere a chi non ha ancora le conoscenze del linguaggio di programmazione C di poter affrontare una parte dei problemi presenti alle selezioni scolastiche. Va quindi vista come una specie di piccolo dizionario del linguaggio: allo stesso modo di un dizionario di una lingua straniera, permette a un neofita di comprendere alcune frasi semplici, ma in generale non permette di comprendere costruzioni complesse né tantomeno di creare dei buoni testi in maniera autonoma.

Inoltre, per non appesantire troppo la lettura, sono state tralasciate caratteristiche “avanzate” del linguaggio e a volte si è ricorso a delle semplificazioni a favore della chiarezza. Nonostante ciò speriamo che possa aiutare nella soluzione dei problemi più semplici e stimoli la curiosità di chi vorrà approfondire la conoscenza di questo linguaggio seguendo i riferimenti riportati alla fine.

Variabili, tipi ed espressioni

In ogni linguaggio di programmazione esiste il concetto di **variabile**, cioè di una zona dove memorizzare dei valori che poi verranno usati per essere elaborati e produrre un risultato. Ogni variabile è identificata da un **nome**, che serve per riferirsi ad essa, e da un **tipo**, che indica che dati può contenere (numeri interi, numeri con la virgola, lettere, ecc.).

Ogni variabile, prima di essere usata, deve essere dichiarata, cioè nel programma va comunicato che tale variabile esiste. Vediamo alcuni esempi di dichiarazioni e i tipi di dato più comuni:

<code>int a;</code>	Variabile di nome <code>a</code> e di tipo intero (può contenere numeri interi)
<code>float b;</code>	Variabile di nome <code>b</code> e di tipo reale (può contenere numeri "con virgola")
<code>double c;</code>	Variabile di nome <code>c</code> e di tipo reale (come float, ma con più decimali)
<code>char d;</code>	Variabile di nome <code>d</code> e di tipo carattere (può contenere un carattere, cioè lettere, cifre o altri segni tipografici)

Per manipolare le variabili esistono poi degli operatori che, combinati con le variabili stesse, producono delle *espressioni*, come quelle a cui si è abituati solitamente in matematica. Alcuni operatori coincidono o corrispondono a quelli ben noti della matematica: ad esempio `+`, `-`, `*` (moltiplicazione) e `/` (divisione). Fa differenza l'operatore `=`, chiamato operatore di **assegnamento**, che serve a inserire un valore in una variabile distruggendo il valore precedentemente contenuto. Richiede che alla sua sinistra ci sia una variabile e alla sua destra ci sia un'espressione (che può essere anche una variabile o una costante) il cui valore verrà poi copiato nella variabile a sinistra.

In effetti l'operazione di assegnamento concluso con il punto e virgola:

variabile = espressione;

è il tipo più frequente di **istruzione** presente nei programmi C.

Esempio

```
int contatore = 0;
int sum = 13;
contatore = contatore + 1;
sum = sum + contatore;
```

Nell'esempio sono presenti due variabili, i cui nomi sono `contatore` e `sum` e il cui tipo per entrambe è intero. Come prima cosa le variabili vengono inizializzate², `contatore` a 0 e `sum` a 13, cioè assumono il valore 0 e 13 rispettivamente, tramite l'operatore di assegnamento. Poi la variabile `contatore` viene aumentata di 1, assumendo quindi il valore 1 e successivamente `sum` assumerà il valore 14 perché viene sommato il proprio valore (in quel momento 13) con il valore di `contatore` (in quel momento 1) e quindi il nuovo valore di `sum` sarà 14.

Gli operatori più comuni sono quelli aritmetici, con l'unica differenza che l'operatore di prodotto non viene indicato con una `x` come si è soliti fare in matematica, ma attraverso il simbolo `*`. Va menzionato il simbolo `%`, che viene usato tra operandi interi, ad esempio in questo modo:

`a = b % c`

e va letto come "calcola il resto (o **modulo**) della divisione intera tra `b` e `c` e assegna il risultato ad `a`".

Esempio

```
int a, b = 5, c = 3;
a = b % c;
```

Il risultato memorizzato in `a` è 2

² L'inizializzazione non è strettamente necessaria: dipende da come verrà utilizzata successivamente la variabile.

Altri operatori comuni solo quelli di **confronto** o comparazione, cioè quelli che vengono solitamente usati in matematica nelle disequazioni (maggiore, minore, ...). Vengono usati per verificare la verità o meno di condizioni imposte dal programmatore: l'uso specifico si vedrà nei prossimi paragrafi, qui segue solo un elenco.

Operatori di confronto

>	Maggiore
<	Minore
>=	Maggiore o uguale
<=	Minore o uguale
==	Uguale
!=	Diverso

Legati sempre alla verifica di condizioni esistono gli *operatori logici* o booleani, che permettono di combinare due o più condizioni per stabilire il valore di verità dell'espressione risultante. Da notare il fatto che in C non esistono delle variabili booleane vere e proprie, ma viene considerato **falso** tutto ciò che vale 0 e **vero** qualsiasi altro valore.

Operatori logici

	OR logico	Operatore binario
&&	AND logico	Operatore binario
!	NOT logico (negazione)	Operatore unario

Un operatore *binario* è un operatore che richiede due operandi. Uno *unario* ne richiede uno solo.

Espressioni idiomatiche

Esistono poi delle espressioni idiomatiche che è bene conoscere perché possono comparire in alcuni problemi.

<code>a++;</code>	La variabile <i>a</i> (intera) è incrementata di una unità: è equivalente a <code>a = a + 1;</code> Si può fare la stessa cosa con <code>--</code> e ovviamente si ha un decremento.
<code>++a;</code>	Produce lo stesso risultato dell'espressione della riga sopra ma cambia l'ordine di esecuzione dell'operazione rispetto ad un eventuale utilizzo del risultato; cioè <i>prima</i> viene fatto l'incremento e <i>dopo</i> viene usato il risultato dell'incremento (vedi esempio sottostante)
<code>a += espr;</code>	Assegna ad <i>a</i> il valore di <i>a</i> stessa sommato con il valore di <i>espr</i> , dove <i>espr</i> è una qualsiasi espressione; è equivalente a: <code>a = a + espr;</code> Si può fare la stessa cosa con tutti gli operatori matematici (<code>-</code> , <code>*</code> , <code>/</code> , <code>%</code>)

Per comprendere meglio la differenza tra l'istruzione `a++` e `++a` vengono proposti degli esempi, in cui si suppone che *a* valga inizialmente 0.

Esempi

<code>b = a++;</code>	<i>a</i> assumerà il valore 1 e <i>b</i> assumerà il valore 0
<code>b = ++a;</code>	<i>a</i> assumerà il valore 1 e <i>b</i> assumerà il valore 1

La valutazione delle espressioni

La valutazione di un'espressione è banale quando sono presenti solo due operandi, diventa più complessa quando sono più di due. Limitandosi ai casi più semplici presenti nei problemi si può dire che gli operatori aritmetici hanno la **precedenza**: fra di loro $*$, $/$ e $\%$ hanno la precedenza su $+$ e $-$. Seguono quelli di confronto e infine quelli logici. Inoltre per tutti gli operatori binari si deve considerare l'**associatività**, cioè come vanno valutate espressioni come $a \text{ op } b \text{ op } c$: si procede *da sinistra verso destra*. Nello stesso modo in cui si usa in matematica, le parentesi possono modificare l'ordine di valutazione delle espressioni, con l'unica differenza che per questo scopo si usano solo le parentesi tonde (le parentesi graffe e quadre si usano con significati differenti) con la possibilità di poterle annidare le une dentro le altre con l'unico vincolo che ogni parentesi aperta abbia una corrispondente parentesi chiusa.

Per fare qualche esempio supponiamo che le tre variabili a , b e c abbiano rispettivamente i valori 1, 2 e 3.

Esempi

<code>d = a + b * c;</code>	d assumerà il valore 7
<code>d = (a + b) * c;</code>	d assumerà il valore 9
<code>a + b > c</code>	L'espressione è falsa perché 3 (il valore di $a + b$) non è maggiore di 3 (valore di c)

Istruzioni per l'input e l'output

Un programma "vero" necessita generalmente di dati in ingresso (o input) e fornisce dei dati in uscita (o output). Negli esercizi proposti alle selezioni questo avviene nel modo classico tramite l'inserimento di dati da tastiera e mostrando il risultato a video. Per permettere queste due operazioni il C fornisce due istruzioni: la `printf` per scrivere a video e la `scanf` per leggere dati da tastiera.

L'uso prevalente che si fa della `printf` è quello di "stampare" (in realtà visualizzare) stringhe costanti che contengono al loro interno i valori di alcune variabili che si vogliono mostrare all'utente. Per comprendere meglio il funzionamento vengono proposti alcuni esempi in cui si suppone che a sia una variabile intera di valore 15, b una variabile float di valore 4.3 e c il carattere 'g'.

Esempi

<code>printf("a vale %d", a);</code>	Scrive a video: a vale 15
<code>printf("b vale %f e c vale %c", b, c);</code>	Scrive a video: b vale 4.300000 e c vale g
<code>printf("%d + %f = %f", a, b, a + b);</code>	Scrive a video: 15 + 4.300000 = 19.300000

Come si vede dagli esempi, nei punti dove si vogliono stampare i valori contenuti nelle variabili di interesse si inseriscono dei "segnaposto" opportuni e in maniera corrispondente, alla fine della stringa, si elencano le variabili di cui si vuole stampare il valore. Per gli scopi di questa guida gli indicatori utili sono `%d` per gli interi, `%f` per i float, `%c` per i caratteri. A volte nella stringa compare la sequenza `\n` che ha il solo scopo di mandare a capo il testo stampato successivamente e quindi è ininfluente ai fini della soluzione dei problemi delle selezioni scolastiche.

In maniera simile la `scanf` permette di assegnare valori alle variabili facendoli inserire dall'utente mediante la tastiera. Anche in questo caso degli esempi possono aiutare.

Esempi

<code>scanf("%d", &a);</code>	in a verrà inserito il numero intero digitato da tastiera
<code>scanf("%f", &b);</code>	in b verrà inserito il numero reale digitato da tastiera

Si osservi l'uso dell'operatore `&` davanti alle variabili. Più avanti ne illustreremo il significato. Vediamo infine un esempio più articolato per comprendere meglio l'uso di `scanf` e `printf`.

Esempio

```
int r;
printf("Inserisci un numero intero compreso fra -10000 e 10000: ");
scanf("%d",&r);
r = r * 3;
printf("La variabile r vale %d\n",r);
```

La prima `printf` ha il solo scopo di illustrare all'ipotetico utilizzatore del programma cosa deve fare; la `scanf` che segue acquisisce il numero che l'utente decide di inserire da tastiera. Supponendo che l'utente abbia deciso di inserire il numero 325 la seconda `printf` scriverà a video la frase: `La variabile r vale 975`.

Istruzioni per il controllo del flusso

Nella costruzione dei programmi si ha la necessità di disporre di **costrutti di controllo** (istruzioni) che permettano di "dirigere" il flusso di esecuzione in base al verificarsi o meno di certe condizioni.

Istruzioni di selezione e blocchi

Queste istruzioni hanno lo scopo di suddividere il programma tra vari rami di esecuzione a seconda del verificarsi o meno di particolari condizioni imposte dal programmatore. Nella versione più semplice questo costrutto prevede, tramite la parola chiave `if`, di verificare se una condizione è vera o falsa e di eseguire il corpo della selezione nel primo caso e saltarlo nel secondo.

Esempio

```
if (c <= r)
    s = s + 1;
```

In questo esempio si verifica se il contenuto della variabile `c` è minore o uguale a quello di `r`, in caso affermativo la variabile `s` verrà incrementata di 1, altrimenti manterrà il valore che aveva in precedenza. Nel caso si volesse condizionare l'esecuzione di più di un'istruzione sarà sufficiente racchiuderle tra parentesi graffe, in modo che vengano considerate un solo *blocco* e vengano eseguite oppure saltate insieme. Un *blocco* può contenere una o più istruzioni e può contenere altri blocchi al suo interno.

Esempio

```
if (c <= r) {
    s = s + 1;
    t = t + 1;
}
```

In questo caso le variabili `s` e `t` vengono entrambe incrementate di 1 se la condizione è vera, altrimenti mantengono il proprio valore.

Un caso più generale è quello che si presenta quando si vuole eseguire una serie di istruzioni oppure un'altra al verificarsi di una condizione.

Esempio

```
if (c <= r)
    s = s + 1;
else
    s = s - 1;
```

In questo caso se il contenuto della variabile `c` è minore o uguale a quello di `r`, `s` viene incrementata di 1, altrimenti viene decrementata di 1. Rispetto a quanto visto prima, la differenza risiede nella possibilità di poter fare qualcosa quando la condizione è falsa, mentre nel primo esempio se la condizione risultava falsa non veniva eseguito alcunché.

Una generalizzazione di quest'ultima forma prevede che dopo l'`else` si possa "attaccare" un nuovo ramo contenente un `if` con eventualmente il proprio `else`.

Esempio

```
if (c < r)
    s = s + 1;
else if (c > r)
    s = s - 1;
else
    s = s + c;
```

Come estensione naturale di questa idea si possono attaccare altri rami `if`, ognuno di seguito ad un `else`, anche se nella pratica situazioni di questo tipo non si verificano di frequente.

Istruzioni di iterazione

Queste istruzioni hanno lo scopo di permettere la ripetizione (“iterazione”) di un blocco di istruzioni, evitando così la scrittura di una sequenza di istruzioni quasi identiche. Per fare un semplice esempio, si supponga di voler stampare la tabellina del 3: usando solo una sequenza di istruzioni ci si troverebbe a dover scrivere 10 volte un’istruzione praticamente uguale, mentre potendo usare istruzioni per l’ iterazione (chiamate anche *cicli*) è sufficiente scrivere del codice che ripeta 10 volte la stampa di una variabile, avendo cura di aumentarla di 3 ad ogni iterazione.

Nel C si trovano tre diversi costrutti di iterazione, sebbene ne sarebbe bastato uno solo: è quindi utile guardarli brevemente per prendere confidenza con ognuno di essi.

L’istruzione *while*

Il ciclo realizzato tramite `while` permette di ripetere il blocco che compare sotto la condizione “mentre” la condizione risulta vera. Riprendendo l’esempio della tabellina del 3 si otterrà il seguente codice.

Esempio

```
while (i <= 10) {
    printf("3 * %d = %d\n", i, 3 * i);
    i++;
}
```

Se si suppone che la variabile `i` abbia come valore iniziale 1, questo frammento di programma stamperà in colonna la tabellina del 3: infatti ad ogni “giro” del `while` il valore di `i` verrà incrementato di 1, permettendo la stampa del nuovo valore della tabellina. Il ciclo terminerà quando `i` avrà il valore 11; infatti, in quel momento la condizione diventerà falsa e il corpo del ciclo non verrà più eseguito. Se la condizione fosse subito falsa, se ad esempio `i` valesse 100, il corpo del ciclo non verrebbe mai eseguito.

L’istruzione *do-while*

Un ciclo molto simile al precedente è quello che solitamente viene chiamato `do-while`, perché presenta la parola `do` prima del corpo del ciclo e la parola `while` al termine: in questo caso la condizione viene verificata alla fine di ogni ciclo per permettere o meno l’esecuzione di un nuovo ciclo.

Esempio

```
do {
    printf("3 * %d = %d\n", i, 3 * i);
    i++;
} while (i <= 10);
```

Come si può vedere l’esempio risulta molto simile al precedente, anche graficamente, e nel caso il valore di partenza di `i` sia 1, si otterrà lo stesso risultato. Nel caso però in cui `i` valesse inizialmente 100 il corpo verrà comunque eseguito una volta, portando alla scrittura di “3 * 100 = 300”, che in questo caso non ha un preciso significato. Questo poiché il controllo sulla condizione viene fatto dopo l’esecuzione del corpo e non prima come nel `while`. Si tenga quindi presente che il corpo del `do-while` viene eseguito almeno una volta.

L'istruzione `for`

Il costrutto `for` è di fatto equivalente a un ciclo `while` opportuno, ma è stato creato per gestire in maniera più comoda e sicura alcuni casi di ciclo che si incontrano molto di frequente nella programmazione. Riprendendo il solito esempio si ha:

Esempio

```
for (i = 0; i < 10; i++)
    printf("3 * %d = %d\n", i + 1, 3 * (i + 1));
```

Come si vede in questo caso il codice risulta più compatto e "naturale", grazie alla sintassi del `for`, che all'interno delle parentesi tonde prevede la presenza di tre sezioni separate da punti e virgola. In questo esempio la prima sezione, solitamente un'inizializzazione, pone il valore di `i` a 0, ed è anche la prima istruzione che viene fatta una volta sola prima che parta il ciclo. Nella seconda sezione, dopo il primo punto e virgola, è presente una condizione, in questo caso `i < 10`, che viene testata prima di iniziare un nuovo ciclo: se risulta vera verrà eseguito il corpo del ciclo, altrimenti il `for` terminerà la sua esecuzione. Infine, nella terza sezione, solitamente è presente un incremento (o un decremento) che in questo caso si trova nella forma idiomatica `i++` e che viene eseguito alla fine di ogni ciclo, dopo l'esecuzione del corpo del `for`. Da notare che in questo esempio si è preferito fare partire il valore di `i` da 0 e non da 1 perché è la forma più comune di inizializzazione.

Se si vuole fare un conteggio in ordine decrescente basta usare `--` invece di `++` nella terza sezione del `for`:

Esempio

```
for (i = 9; i >= 0 ; i--)
    printf("3 * %d = %d\n", i + 1, 3 * (i + 1));
```

Vettori

I **vettori** sono dei tipi particolari di variabile che nei linguaggi di programmazione hanno lo scopo di gestire un certo numero di variabili dello stesso tipo senza doverle dichiarare una a una e utilizzarle in maniera singola. Se ad esempio si volessero stampare dal più piccolo al più grande 10 numeri inseriti in ordine casuale dall'utente, si avrebbe la necessità di memorizzarli, applicare un qualche metodo per ordinarli e successivamente visualizzarli. Risulta evidente che se dovessimo chiamare ogni variabile singolarmente ci si troverebbe con 10 variabili e la stesura del codice per l'ordinamento prevederebbe un numero notevolissimo di righe per realizzare un programma piuttosto semplice. Se poi i numeri inseriti fossero migliaia o milioni è facile immaginare che sarebbe praticamente impossibile scrivere il programma.

L'idea che risolve tutti i problemi di questo tipo parte dall'osservazione che l'unico elemento che differenzia una variabile dall'altra è la sua posizione, cioè ci si potrebbe riferire a ogni variabile con degli ordinali, indicandole come la *prima*, la *seconda*, la *terza* e così via: in questo modo tutte le variabili avrebbero lo stesso nome, `v` ad esempio, ma ognuna verrebbe distinta dalle altre attraverso un *indice*, cioè un intero positivo che ne indica la posizione. Potremmo così avere `v1`, `v2`, `v3` e così via. La sintassi del C prevede che per rappresentare questa idea si scriva una variabile di tipo vettore seguita da una coppia di parentesi quadre, all'interno delle quali inserire l'indice che individua la variabile specifica (detta *elemento* del vettore).

Esempio

```
int v[10], n[20];
v[3] = 7;
n[5] = v[3];
```

In questo esempio vengono prima dichiarati due vettori, il vettore `v` che può contenere 10 elementi e il vettore `n` che ne può contenere 20, entrambi di tipo intero: la dichiarazione della dimensione dei vettori è necessaria per motivi tecnici, ma nei problemi delle scolastiche ha poca importanza perché compare già nel codice. Successivamente il quarto elemento del vettore `v` (il quarto perché in C gli indici dei vettori partono da 0 e non da 1) viene ad assumere il valore 7 e successivamente il sesto elemento del vettore `n` assume il

valore del quarto elemento del vettore `v`, cioè ancora 7. Tutti gli altri elementi di entrambi i vettori risultano avere valori indefiniti³.

Il seguente esempio è invece un pezzo di codice più realistico che mostra un possibile utilizzo dei vettori.

Esempio

```
int M[10],i;
for (i = 0;i < 10; i++)
    scanf("%d", &M[i]);
for (i = 9;i >= 0; i--)
    if (i % 2 == 0) printf ("%d", M[i]);
```

Dopo aver dichiarato un vettore `M` di lunghezza 10 elementi, il primo ciclo `for` fa inserire all'utente 10 valori nelle 10 posizioni del vettore. Il secondo ciclo stampa solo i valori nelle posizioni di indice pari (la condizione è vera solo se la divisione di `i` per 2 ha resto 0, che è equivalente a dire che `i` è pari) in ordine inverso a quello di inserimento.

Un tipo particolare di vettore è quello formato da caratteri, solitamente indicato come *stringa*. La sua importanza deriva dal fatto che praticamente ogni programma ha la necessità di lavorare con le stringhe, per visualizzarle, elaborarle, modificarle, ma nel contesto dei problemi delle selezioni possono essere immaginate alla stregua dei vettori di altri tipi.

Matrici

Una naturale estensione dei vettori sono le **matrici**, a volte dette vettori bidimensionali (o più in generale multidimensionali) perché al posto di avere un solo indice ne hanno due o più. Il modo più semplice di immaginare una matrice bidimensionale è di pensare a una griglia rettangolare in cui ogni elemento è individuato da una posizione sulle righe e da una sulle colonne, come nel noto gioco battaglia navale, nel quale la forma quadrata della griglia è solo un caso particolare della forma rettangolare. Supponendo di avere una matrice `M` con 30 righe e 24 colonne, in cui ogni valore in posizione `M[i][j]` rappresenta la temperatura misurata il giorno `i` del mese di giugno all'ora `j` di quel giorno, è facile stampare le medie giornaliere delle temperature con il seguente codice.

Esempio

```
float M[30][24], somma;
int i, j;
...
for (i = 0; i < 30; i++){
    somma = 0;
    for (j = 0; j < 24; j++)
        somma += M[i][j];
    printf ("La media del giorno %d è stata di %f gradi\n", i + 1, somma/24);
}
```

Come si può vedere la dichiarazione di una matrice è una naturale estensione di quella di un vettore: dopo aver acquisito i dati (nel codice compaiono dei puntini per brevità) il `for` più esterno ha lo scopo di scorrere i 30 giorni in cui i dati sono acquisiti e il `for` interno aggiunge i valori giornalieri nella variabile `somma`. Da notare che la variabile `somma` viene riportata a 0 ad ogni nuovo ciclo del `for` esterno, altrimenti i valori di temperatura si accumulerebbero e solo il primo giorno avrebbe la media giusta.

Funzioni

I *sottoprogrammi* sono una delle modalità che offrono i linguaggi per poter organizzare meglio il codice e per poter esprimere la ricorsione (come si vedrà nel prossimo paragrafo). In C i sottoprogrammi sono chiamati *funzioni*.

Per chiarirne gli scopi vediamo un esempio preso dal mondo culinario: supponiamo di trovarci nella cucina di un ristorante e di essere il cuoco. Come cuoco conosceremo una serie di ricette, ognuna definita in modo da permettere di realizzare un certo piatto. Quando la sala inizia a riempirsi e arrivano le ordinazioni, il

³ In realtà il comportamento delle posizioni non inizializzate esplicitamente dipende dal fatto che il vettore sia globale o locale, ma ai fini di questa Guida ciò è ininfluenza.

cameriere chiama i piatti dicendoci cose del tipo “4 trofie con il pesto” oppure “3 bistecche di cui 2 al sangue e una a cottura media”. In quel momento le nostre ricette prendono vita e si trasformano in piatti da portare ai tavoli. In questa analogia ogni ricetta è una funzione, cioè una serie di istruzioni per realizzare un compito, che però non fa niente finché non viene “chiamata” dal cameriere. Quando arriva l’ordine non solo viene chiamata la ricetta, ad esempio “trofie al pesto”, ma si indicano anche alcuni parametri, per la prima ricetta quanti piatti se ne devono fare, per la seconda sia la quantità che il grado di cottura.

Ricapitolando ogni funzione in C è individuata da un nome, contiene una serie di istruzioni per eseguire un compito (fare calcoli, analizzare stringhe, ecc.) e può essere chiamata passandogli dei parametri. La sintassi di una funzione che prende due numeri interi e ne restituisce la somma potrebbe essere questa:

Esempio

```
int somma (int a, int b)
{
    int c = a + b;
    return c;
}
```

Nell’esempio si vede la “ricetta” della somma, racchiusa tra parentesi graffe, che ne delimitano il corpo: i parametri sono le due variabili *a* e *b*, viene creata una variabile *c* per contenere il risultato della somma, il cui valore viene restituito (o come impropriamente si è abituati a dire “ritornato”) al chiamante : è come se il cuoco, dopo aver fatto i piatti richiesti, li desse al cameriere per portarli a tavola.

Esempio

```
main()
{
    int r, s, t;
    scanf("%d", &r);
    scanf("%d", &s);
    t = somma(r, s);
    printf("%d", t);
}
```

In questo esempio si può vedere una funzione particolare, il *main*, che compare in qualsiasi programma reale (mentre non è detto che sia sempre presente negli esercizi delle scolastiche) il cui scopo è indicare da dove deve iniziare il programma: in un scenario in cui un programma contenesse molte funzioni è necessario istruire il computer da quale di queste deve partire e questa è sempre il *main*. In questo brevissimo programma vengono prima inseriti due numeri dall’utente, poi viene chiamata la funzione *somma* definita nell’esempio precedente, passandogli come parametri i numeri appena letti nell’ordine in cui compaiono nella chiamata. La funzione esegue il proprio codice e restituisce il valore calcolato, che verrà quindi copiato in *t* per poi essere visualizzato.

Esistono anche funzioni che non restituiscono alcun valore, che in alcuni linguaggi vengono chiamate *procedure*: in C non esiste una parola chiave specifica per indicarle, ma è sufficiente mettere al posto delle parole chiave usate per indicare un valore di ritorno di un tipo specifico (*int*, *float*, ...) la parola chiave *void*, che indica appunto l’assenza di un valore di ritorno. Avendo in mente la funzione *somma* ci si potrebbe domandare a cosa serve una procedura: nonostante non producano valori di ritorno le procedure possono interagire con l’ambiente di esecuzione, ad esempio visualizzando scritte o in altri modi. Tornando al paragone con il ristorante è come se il cameriere tornasse in cucina con i complimenti fatti da un cliente al cuoco: a differenza di un ordine poi il cameriere non porterà niente in sala (nessun valore di ritorno) ma comunque avrà un effetto sull’umore della cucina...

Esempio

```
void complimenti(int n)
{
    int i;
    for (i=0; i<n; i++)
        printf("Complimenti al cuoco");
}
```

Il parametro *n* in questo caso viene usato per indicare quante volte si vogliono ripetere i complimenti.

Da notare comunque che a causa della mancanza di un valore di ritorno non è possibile inserire una procedura in un contesto dove è necessario un valore, ad esempio in un assegnamento.

Esempio

```
main()
{
    int a = complimenti(3);      sbagliato
    int b = a + complimenti(3); sbagliato
    complimenti(3);             corretto
}
```

Funzioni ricorsive

Un tipo particolare di funzioni sono quelle *ricorsive*, quelle cioè che nelle proprie istruzioni contengono delle chiamate a sé stesse. Anche se a prima vista questo può sembrare strano è del tutto legittimo e permette, in molti casi concreti, di risolvere in maniera elegante problemi altrimenti di difficile soluzione. Anche in questo caso può essere d'aiuto un esempio⁴.

Esempio

```
int prodotto(int a, int b)
{
    if (a == 0 || b == 0) return 0;
    return b + prodotto(a - 1, b);
}
```

Lo scopo di questa funzione, come si evince dal nome, è restituire il prodotto tra i due parametri interi ed è sicuramente ricorsiva, perché richiama sé stessa nella seconda riga del suo codice. Caratteristica di tutte le funzioni ricorsive è di avere una o più condizioni di uscita, altrimenti la funzione continuerebbe a chiamare sé stessa senza mai finire: in questo caso la prima riga verifica se *a* o *b* sono zero e in quel caso termina la ricorsione. Se né *a* né *b* sono zero viene implementata l'idea di poter ottenere il prodotto come una serie di somme, in questo caso sommando *b* a sé stesso *a* volte. Di fatto quello che viene applicato è l'identità matematica :

$$a * b = (a - 1) * b + b$$

Provando a sostituire ad *a* e *b* due valori e svolgendo la funzione fino a quando non ritorna si può vedere che si ottiene il risultato corretto. Utilizziamo i valori *a* = 3 e *b* = 2 per verificare tramite una tabella quello che succede.

<i>a</i>	<i>b</i>	Chiamata	Condizione $a == 0 \ \ b == 0$	Valore di ritorno
3	2	prodotto(3,2)	falsa	2 + prodotto(2,2)
2	2	prodotto(2,2)	falsa	2 + prodotto(1,2)
1	2	prodotto(1,2)	falsa	2 + prodotto(0,2)
0	2	prodotto(0,2)	vera	0

Se si sostituiscono a ritroso i valori di ritorno si vede che il risultato è proprio 6.

Ma funzionerà sempre questo codice? Questa implementazione richiede che *a* sia maggiore o uguale a zero, altrimenti si entrerà in una ricorsione infinita, che nella realtà porta alla terminazione "forzata" del programma.

Anche le funzioni `void` (procedure) possono essere ricorsive. Ad esempio la versione ricorsiva di `complimenti` introdotta nel paragrafo precedente diventerebbe:

⁴ Questo codice serve solo per fornire un semplice esempio, nella realtà nessun programmatore definirebbe il prodotto in questo modo, userebbe semplicemente l'operatore `*`.

```

void complimenti(int n)
{
    if (n>0)
    {
        printf("Complimenti al cuoco");
        complimenti(n-1);
    }
}

```

Passaggio di parametri per riferimento

Negli esempi visti finora i valori dei parametri sono passati *per valore* (o *per copia*) che è il metodo normale di passaggio dei parametri. Questo vuol dire che se vengono passate delle variabili in realtà vengono copiati solo i loro valori nei parametri e le variabili stesse mantengono il valore che avevano prima della chiamata della funzione.

Viceversa in quello che chiameremo impropriamente⁵ *passaggio per riferimento*, le variabili passate alla funzione chiamata, se modificate all'interno di essa, risultano modificate anche nel processo chiamante.

Un esempio classico per vedere la differenza tra passaggio per valore e per riferimento è il seguente:

Esempio

```

void scambio_valore(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

void scambio_riferimento(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

main()
{
    int c = 7, d = 9;
    scambio_valore(c, d);
    printf("%d %d \n", c, d);
    scambio_riferimento(&c, &d);
    printf("%d %d \n", c, d);
}

```

Sebbene le due funzioni (in questo caso sarebbe più preciso chiamarle procedure) sembrano molto simili, il loro comportamento è decisamente diverso. La prima funzione, in cui i parametri vengono passati per valore, scambia le copie di `c` e `d`, quindi la `printf` che la segue stamperà 7 e 9, poiché le variabili `c` e `d` non sono state modificate. Invece la funzione `scambio_riferimento` accetta il passaggio dei parametri per riferimento, cosa che viene indicata dalla presenza degli asterischi prima dei nomi di variabile. Quando nel corpo della funzione ci si riferisce a `*a` e `*b` si intende che si lavora direttamente sui parametri passati (in questo caso `c` e `d`) e non su delle loro copie. Quindi questo comporterà che la seconda `printf` stamperà 9 e 7, poiché i valori contenuti in `c` e `d` sono stati scambiati dalla funzione. Da notare infine che quando viene usato il passaggio per riferimento le variabili passate devono avere il simbolo `&` (*operatore di indirizzo*) prima del nome. Si rinviano gli studenti interessati ad approfondire l'utilizzo di questo operatore nelle fonti indicate in fondo alla guida, considerando che questo aspetto e tutti quelli collegati sono comunque tra i più ostici del linguaggio, soprattutto quando si sia interessati a capire il perché e il quando vengano usati.

Questa potenzialità del linguaggio viene utilizzata anche quando si vogliono passare a una funzione un vettore o una matrice, sempre per motivi tecnici legati alle caratteristiche del linguaggio C.

⁵ In C non esiste un passaggio di riferimento vero e proprio, ma si usa il meccanismo del passaggio di indirizzo per ottenere lo stesso risultato. Per approfondimenti si vedano i testi citati nella bibliografia.

Nel caso di passaggio di un vettore, supponendo di voler disporre di una funzione che sommi i primi n elementi, si potrebbe esprimerla in questo modo:

Esempio

```
int somma(int v[], int n)
{
    int i, s = 0;
    for (i = 0; i < n; i++)
        s += v[i];
    return s;
}
```

La stessa funzione potrebbe anche essere scritta in questo modo:

```
int somma(int *v, int n)
```

e il corpo sarebbe il medesimo. Quando si usano i vettori è comune trovare la prima notazione; ma in alcuni casi anche la seconda viene usata e quindi vale la pena conoscerle entrambe.

Volendo poi usare la funzione in un programma e supponendo di avere un vettore di nome Q e contenente 10 elementi si potrebbe fare così:

```
int ris = somma(Q , 10);
```

dove in ris verrebbe inserito il valore della somma dei dieci elementi del vettore Q . In questo caso, siccome Q è un vettore, non viene usato il simbolo $\&$ prima del nome, sempre a causa della sintassi del C.

Si intende che del vettore viene passato solo l'indirizzo (non viene cioè creata una copia del vettore). Così, per azzerare i primi n elementi di un vettore potremmo usare la procedura.

```
void azzerare(int v[], int n)
{
    int i;
    for (i = 0; i < n; i++) v[i] = 0;
}
```

Un esempio di chiamata potrebbe essere:

```
azzerare(Q,10);
```

Passaggio di matrici

Nel passaggio di una matrice si ha una sintassi simile a quella del passaggio dei vettori, con le modifiche richieste dal C per l'aggiunta di un'ulteriore dimensione.

Esempio

```
int somma(int v[][10], int n)
{
    int i, j, s = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < 10; j++)
            s += v[i][j];
    return s;
}
```

Come si può vedere, nella dichiarazione dei parametri *deve essere specificato* il numero di colonne della matrice (in questo caso 10).

Direttive al preprocessore

Nei programmi C a volte si trovano delle istruzioni particolari che iniziano con il segno di cancelletto (#). Queste istruzioni non fanno parte del linguaggio C e tecnicamente sono definite *direttive al preprocessore* (ovvero al compilatore che traduce il programma in istruzioni eseguibili dalla macchina): segue una brevissima spiegazione delle due che si possono incontrare negli esercizi.

#include

Questa direttiva indica al preprocessore di includere un file, ad esempio `stdio.h`, che serve al compilatore per verificare la correttezza nell'uso delle funzioni di libreria.

Esempio

```
#include <stdio.h>
#include <stdlib.h>
```

Ai fini della soluzione degli esercizi si può ignorare. Tuttavia un programma per essere eseguito richiede solitamente almeno la prima direttiva.

Esempio

```
#include <stdio.h>
int main()
{
    printf("Salve\n");
    return 0;
}
```

#define

Questa direttiva indica al preprocessore di sostituire testualmente ogni occorrenza del primo valore indicato con il secondo, può essere vista come una specie di "Trova/Sostituisci".

Esempio

```
#define D 10
#define MAX 10000
```

In questo esempio il preprocessore sostituirà ogni occorrenza del simbolo `D` maiuscolo isolato con il valore 10 e ogni occorrenza della scritta `MAX` con 10000.

È la direttiva usata per definire le cosiddette costanti: valori che non variano durante l'esecuzione di un programma ma che si vuole poter facilmente ridefinire senza dover rianalizzare e modificare l'intero programma. In questo esempio, si può pensare che `D` sia la dimensione di un vettore: nel collaudo del programma, si usa un vettore di piccole dimensioni (10 elementi) per verificarne rapidamente il riempimento, ma si vuole poter ampliare facilmente le sue dimensioni (per esempio portandole a 1000 elementi) una volta terminato il collaudo del programma.

Riferimenti per approfondimenti

Esistono molti libri e manuali sul C. Per chi volesse iniziare a prendere confidenza con il linguaggio sono consigliati i seguenti testi:

Il linguaggio C. Principi di programmazione e manuale di riferimento (2° edizione 1 gennaio 2004)

Autori: Brian W. Kernighan, Dennis M. Ritchie.

Editore: Pearson.

Testo di riferimento sul linguaggio, è stato il primo manuale del C, scritto dall'autore del linguaggio. Anche se richiede la conoscenza di alcuni concetti basilari di programmazione, può essere usato come manuale introduttivo del linguaggio.

C. Corso completo di programmazione (4° edizione 12 novembre 2010)

Autori: Paul J. Deitel, Harvey M. Deitel

Editore: Apogeo.

Questo testo è sicuramente più didattico, parte dalle basi e insegna a programmare utilizzando il C come strumento per spiegare concetti generali. Adatto anche a chi non ha mai programmato.

Per migliorare la conoscenza e per rinforzare i concetti esposti in questo tutorial è importante poter provare e apportare modifiche ai programmi che si trovano nei testi delle gare scolastiche degli scorsi anni.

In rete si trovano diversi ambienti di sviluppo gratuiti per C, come quelli suggeriti nel sito delle olimpiadi di informatica:

<http://www.olimpiadi-informatica.it/>

visitando la sezione dedicata al software.

Tuttavia, chi non vuole o non può installare questi ambienti può ugualmente provare/modificare i programmi che compaiono collegandosi a siti che permettono l'editing, la compilazione e l'esecuzione di programmi come, ad esempio:

<http://www.compileonline.com>

o anche:

<http://www.ideone.com/>

e lavorando di copia, incolla e completa.