Lo scopo di questo corso di BASIC su disco e' quello di metterti in grado di capire ed eseguire programmi in quel linguaggio. L'istruzione parte da zero, e poco alla volta insegna a programmare. Non si tratta quindi di un corso per 'addetti ai lavori', ma di qualcosa concepito espressamente per chi e' alle primissime armi.
Si e' cercato di fare un corso in autoistruzione il piu' completo e vario possibile. Siamo convinti che anche coloro che hanno un po' di pratica col BASIC possano trarre giovamento da queste lezioni: la materia e' vasta e non si finisce mai di imparare.

A quanto ci consta, non esiste attualmente niente di simile sul mercato: i pochi tentativi reperibili sono ridotti ai minimi termini e sono di qualche aiuto solo a chi ha gia' una notevole conoscenza del linguaggio BASIC. Siamo insomma convinti di offrire un prodotto d'eccezione, il cui allestimento non e' stato dei piu' semplici.

Su questi dischetti sono incise 45 lezioni che trattano poco alla volta i vari concetti del linguaggio BASIC. Ti consigliamo di seguirle in ordine progressivo per ottenere un profitto soddisfacente ed un grado di apprendimento uniforme.
La stampante ti dara' il testo completo delle varie lezioni. Ciascuna di esse contiene l'esposizione dei concetti fondamentali del BASIC, accompagnati da numerosi esempi pratici. Dovrai rispondere a diverse domande che via via troverai nel testo. Quanto richiesto sara' visualizzato anche dal monitor.
Se la risposta sara' quella giusta, il programma proseguira', altrimenti il quesito ti verra' riproposto dal monitor per un secondo od un terzo tentativo. In ogni caso sul testo stampato comparira' solamente la risposta giusta con la relativa spiegazione.

Questo metodo si prefigge due scopi: 1) metterti in grado di capire da solo il livello di apprendimento che via via avrai raggiunto nei vari argomenti; 2) fornirti un vero e proprio testo di studio sulla programmazione in BASIC, da conservare e consultare in caso di dubbi.

La stesura del testo presuppone l'uso di una stampante ad 80 colonne.
Usando lunghezze di riga diverse si verificheranno ritorni a capo di linea che non rispettano le regole grammaticali, ma il programma girera' correttamente in ogni caso. Per ottenere un risultato perfetto consigliamo quindi l'uso di una stampante con 80 caratteri per riga.

La stampa e' realizzata poi in modo tale che, posizionando ad ogni lezione l'inizio del foglio in maniera appropriata, la lezione stessa viene stampata pagina per pagina, con le relative numerazioni e le opportune spaziature tra un foglio e quello successivo. Se userai carta della lunghezza standardizzata di 28 centimetri, potrai separare le varie pagine seguendo la perforazione; ordinandole poi progressivamente ed inserendole negli appositi raccoglitori che si trovano in commercio, otterrai un libro abbastanza voluminoso.
Ricorda poi una cosa importante: quando la stampante si ferma, devi osservare il monitor; troverai le indicazioni per proseguire. Caso per caso si trattera' ad esempio di rispondere ad una domanda, o di premere qualche tasto indicato.

Per poter permettere l'uso delle lezioni di BASIC anche a coloro che dispongono di una memoria centrale non troppo estesa, ogni lezione e' stata divisa in due o tre parti; la seconda e la terza vengono agganciate automaticamente da quella precedente, e tale operazione richiede pochi secondi. In tal modo anche con una disponibilita' di soli 48 K di memoria si possono eseguire le varie lezioni.

Un'ultima cosa: per eseguire realmente i vari esempi che troverai indicati dovrai uscire dal programma e digitare gli esempi stessi, ti avviamo quindi fin

vrai uscire dal programma e digitare gli esempi stessi, ti conviene quindi farlo a lezione terminata, altrimenti dovrai riprenderla dall'inizio.

L E Z I O N I D I B A S I C

A CURA DELLA REDAZIONE DI NUOVA ELETTRONICA

E DELL'ING. ROMANO CICOGNANI

PREFAZIONE

Lo scopo di questo corso di BASIC su disco e' quello di metterti in grado di capire ed eseguire programmi in quel linguaggio. L'istruzione parte da zero, e poco alla volta insegna a programmare. Non si tratta quindi di un corso per 'addetti ai lavori', ma di qualcosa concepito espressamente per chi e' alle primissime armi.
Si e' cercato di fare un corso in autoistruzione il piu' completo e vario possibile. Siamo convinti che anche coloro che hanno un po' di pratica col BASIC possano trarre giovamento da queste lezioni: la materia e' vasta e non si finisce mai di imparare.

A quanto ci consta, non esiste attualmente niente di simile sul mercato: i pochi tentativi reperibili sono ridotti ai minimi termini e sono di qualche aiuto solo a chi ha gia' una notevole conoscenza del linguaggio BASIC. Siamo insomma convinti di offrire un prodotto d'eccezione, il cui allestimento non e' stato dei piu' semplici.

Su questi dischetti sono incise 45 lezioni che trattano poco alla volta i vari concetti del linguaggio BASIC. Ti consigliamo di seguirle in ordine progressivo per ottenere un profitto soddisfacente ed un grado di apprendimento uniforme.
La stampante ti dara' il testo completo delle varie lezioni. Ciascuna di esse contiene l'esposizione dei concetti fondamentali del BASIC, accompagnati da numerosi esempi pratici. Dovrai rispondere a diverse domande che via via troverai nel testo. Quanto richiesto sara' visualizzato anche dal monitor.
Se la risposta sara' quella giusta, il programma proseguira', altrimenti il quesito ti verra' riproposto dal monitor per un secondo od un terzo tentativo. In ogni caso sul testo stampato comparira' solamente la risposta giusta con la relativa spiegazione.

Questo metodo si prefigge due scopi: 1) metterti in grado di capire da solo il livello di apprendimento che via via avrai raggiunto nei vari argomenti; 2) fornirti un vero e proprio testo di studio sulla programmazione in BASIC, da conservare e consultare in caso di dubbi.

La stesura del testo presuppone l'uso di una stampante ad 80 colonne.
Usando lunghezze di riga diverse si verificheranno ritorni a capo di linea che non rispettano le regole grammaticali, ma il programma girera' correttamente in ogni caso. Per ottenere un risultato perfetto consigliamo quindi l'uso di una stampante con 80 caratteri per riga.

La stampa e' realizzata poi in modo tale che, posizionando ad ogni lezione l'inizio del foglio in maniera appropriata, la lezione stessa viene stampata pagina per pagina, con le relative numerazioni e le opportune spaziature tra un foglio e quello successivo. Se userai carta della lunghezza standardizzata di 28 centimetri, potrai separare le varie pagine seguendo la perforazione; ordinandole poi progressivamente ed inserendole negli appositi raccoglitori che si trovano in commercio, otterrai un libro abbastanza voluminoso.

Ricorda poi una cosa importante: quando la stampante si ferma, devi osservare il monitor; troverai le indicazioni per proseguire. Caso per caso si trattera' ad esempio di rispondere ad una domanda, o di premere qualche tasto indicato.

Per poter permettere l'uso delle lezioni di BASIC anche a coloro che dispongono di una memoria centrale non troppo estesa, ogni lezione e' stata divisa in due o tre parti; la seconda e la terza vengono agganciate automaticamente da quella precedente, e tale operazione richiede pochi secondi. In tal modo anche con una disponibilita' di soli 48 K di memoria si possono eseguire le varie lezioni.

Un'ultima cosa: per eseguire realmente i vari esempi che troverai indicati dovrai uscire dal programma e digitare gli esempi stessi; ti conviene quindi farlo a lezione terminata, altrimenti dovrai riprenderla dall'inizio.

INDICE

APPENDICE N. 1 PAROLE CHIAVE

1 - →
2 - !
3 - "
4 - #
5 - $
6 - %
7 - &
8 - &H
9 - &O
10 - '
11 - ( )
12 - (LF)
13 - (,)
14 - (.)
15 - (A)
16 - (C)
17 - (CTRL+A)
18 - (CTRL+H)
19 - (CTRL+I)
20 - (CTRL+J)
21 - (CTRL+L)
22 - (CTRL+P)
23 - (CTRL+R)
24 - (CTRL+T)
25 - (CTRL+X)
26 - (CTRL+Y)
27 - (D)
28 - (DEL)
29 - (E)
30 - (ESC)
31 - (H)
32 - (I)
33 - (II)
34 - (K)
35 - (L)
36 - (Q)
37 - (RETURN)
38 - (S)
39 - (SPAZIO)
40 - (X)
41 - (→)
42 - *
43 - +
44 - ,
45 - -
46 - .
47 - /

```
48  - :
49  - ;
50  - <
51  - <=
52  - <>
53  - =
54  - >
55  - >=
56  - ?
57  - @
58  - ABS
59  - AND
60  - APPEND
61  - ASC
62  - ATN
63  - ATTRIB
64  - AUTO
65  - BASIC
66  - CDBL
67  - CHR$
68  - CINT
69  - CLEAR
70  - CLOSE
71  - CLS
72  - CMD"D"
73  - CMD"E"
74  - CMD"S"
75  - CMD"funzione DOS"
76  - CONT
77  - COPY
78  - COS
79  - CSNG
80  - CVD
81  - CVI
82  - CVS
83  - DATA
84  - DATE
85  - DEBUG
86  - DEFDBL
87  - DEFFN
88  - DEFINT
89  - DEFSNG
90  - DEFSTR
91  - DEFUSR
92  - DELETE (D)
93  - DIM
94  - DIR
95  - DUMP
96  - E
97  - EDIT (E)
98  - END
99  - EOF
```

100 - ERL
101 - ERR
102 - ERROR
103 - EXP
104 - FIELD
105 - FIX
106 - FOR...TO...STEP...NEXT
107 - FORMAT
108 - FRE
109 - FREE
110 - GET
111 - GOSUB
112 - GOTO
113 - IF...THEN...ELSE
114 - INKEY$
115 - INP
116 - INPUT
117 - INPUT#
118 - INSTR
119 - INT
120 - KILL
121 - LEFT$
122 - LEN
123 - LET
124 - LIB
125 - LINEINPUT
126 - LINEINPUT#
127 - LIST (L)
128 - LLIST
129 - LOAD
130 - LOC
131 - LOF
132 - LOG
133 - LPRINT
134 - LSET
135 - MEM
136 - MERGE
137 - MID$
138 - MKD$
139 - MKI$
140 - MKS$
141 - NEW
142 - NOT
143 - ON...GOSUB
144 - ON...GOTO
145 - ONERRORGOTO
146 - OPEN
147 - OR
148 - OUT
149 - PEEK
150 - POINT
151 - POKE

152 - POS
153 - PRINT (?)
154 - PRINT# (?#)
155 - PRINT@ (?@)
156 - PRINTTAB (?TAB)
157 - PRINTUSING (?USING)
158 - PROT
159 - PUT
160 - RANDOM
161 - READ
162 - REF
163 - REM (')
164 - RENAME
165 - RENUM
166 - RESET
167 - RESTORE
168 - RESUME
169 - RETURN
170 - REVOFF
171 - REVON
172 - RIGHT$
173 - RND
174 - RSET
175 - RUN
176 - SAVE
177 - SET
178 - SGN
179 - SIN
180 - SQR
181 - STOP
182 - STR$
183 - STRING$
184 - SYSTEM
185 - TAN
186 - TIME
187 - TIME$
188 - TROFF
189 - TRON
190 - USR
191 - VAL
192 - VARPTR
193 - VERIFY

APPENDICE N.2 CODICI DEGLI ERRORI

CODICE	MESSAGGIO	SPIEGAZIONE
1	NEXT WITHOUT FOR	NEXT senza FOR
2	SYNTAX ERROR	Errore di sintassi
3	RETURN WITHOUT GOSUB	RETURN senza GOSUB
4	OUT OF DATA	Fine DATA
5	ILLEGAL FUNCTION CALL	Funzione illecita
6	OVERFLOW	Numero troppo piccolo o troppo grande
7	OUT OF MEMORY	Manca memoria
8	UNDEFINED LINE	Linea inesistente
9	SUBSCRIPT OUT OF RANGE	L'elemento di matrice va oltre la DIM
10	RIDIMENSIONED ARRAY	Ridimensionamento di matrice
11	DIVISION BY ZERO	Divisione per zero
12	ILLEGAL DIRECT	INPUT come comando diretto
13	TYPE MISMATCH	Numero assegnato ad una stringa o viceversa
14	OUT OF STRING SPACE	Poco spazio per le stringhe
15	STRING TOO LONG	Striga che supera i 255 caratteri
16	STRING FORMULA TOO COMPLEX	Operazione su stringa troppo complessa
17	CAN'T CONTINUE	CONT non puo' essere eseguito
18	NO RESUME	Non e' stato dato RESUME
19	RESUME WITHOUT ERROR	RESUME senza ONERRORGOTO
20	UNPRINTABLE ERROR	Errore non stampabile
21	MISSING OPERAND	Manca l'operando
22	BAD FILE DATA	INPUT di dati non corretto
51	FIELD OVERFLOW	Riservati piu' di 255 bytes con FIELD
52	INTERNAL ERROR	Errore interno oppure di IN/OUT
53	BAD FILE NUMBER	Uso improprio di numero di file
54	FILE NOT FOUND	Nome di file inesistente
55	BAD FILE MODE	Uso errato del file
56	FILE OLREADY OPEN	File gia' aperto
58	DISK I/O ERROR	Errore nella trasmissione dati col disco
59	DUPLICATE FILE NAME	Nome di file gia' esistente
62	DISK FULL	Disco pieno
63	INPUT PAST END	Fine del FILE superata
64	BAD RECORD NUMBER	Numero di record superiore a 340
65	BAD FILE NAME	Nome di file inammissibile
67	DIRECT STATEMENT IN FILE	LOAD-RUN-MERGE eseguiti su un file non BASIC
68	TOO MANY FILES	Piu' di 48 files in un disco

APPENDICE N. 3 CODICI ASCII (32-127)

32	-	
33	-	!
34	-	"
35	-	#
36	-	$
37	-	%
38	-	&
39	-	'
40	-	(
41	-	)
42	-	*
43	-	+
44	-	,
45	-	-
46	-	.
47	-	/
48	-	0
49	-	1
50	-	2
51	-	3
52	-	4
53	-	5
54	-	6
55	-	7
56	-	8
57	-	9
58	-	:
59	-	;
60	-	<
61	-	=
62	-	>
63	-	?
64	-	@
65	-	A
66	-	B
67	-	C
68	-	D
69	-	E
70	-	F
71	-	G
72	-	H
73	-	I
74	-	J
75	-	K
76	-	L
77	-	M
78	-	N
79	-	O
80	-	P
81	-	Q
82	-	R
83	-	S
84	-	T
85	-	U
86	-	V
87	-	W
88	-	X
89	-	Y
90	-	Z
91	-	↑
92	-	↓
93	-	←
94	-	→
95	-	
96	-	`
97	-	a
98	-	b
99	-	c
100	-	d
101	-	e
102	-	f
103	-	g
104	-	h
105	-	i
106	-	j
107	-	k
108	-	l
109	-	m
110	-	n
111	-	o
112	-	p
113	-	q
114	-	r
115	-	s
116	-	t
117	-	u
118	-	v
119	-	w
120	-	x
121	-	y
122	-	z
123	-	{
124	-	|
125	-	}
126	-	~
127	-	

APPENDICE N. 4 MAPPA VIDEO

Una pagina video e' formata da 16 righe di 32 caratteri ciascuna. Il primo carattere della prima riga porta il numero zero; i caratteri successivi hanno numeri via via crescenti, fino ad arrivare all'ultimo carattere della sedicesima riga, che porta il numero 511.
Nella figura seguente, ogni lettera 'O' rappresenta un carattere; all'inizio e alla fine di ogni riga c'e' il numero del carattere d'estremita'.

```
0   OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 31
32  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 63
64  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 95
96  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 127
128 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 159
160 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 191
192 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 223
224 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 255
256 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 287
288 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 319
320 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 351
352 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 383
384 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 415
416 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 447
448 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 479
480 OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO 511
```

L E Z I O N E n. 1

INDICE

1 - INTRODUZIONE

Chiariamo innanzitutto il concetto 'PROGRAMMARE IN BASIC'.

PROGRAMMARE significa fare un elenco di istruzioni che, una volta immesse nel computer, siano da questo capite ed eseguite; cosi' facendo i DATI introdotti vengono elaborati nel modo stabilito dal programma per dare i RISULTATI voluti.

Devi assimilare subito un concetto importante: il computer non e' dotato di intelligenza e neppure di buon senso. E' soltanto in grado di eseguire un certo numero di ISTRUZIONI ben definite: se vengono date come di dovere non segnala errore e le esegue correttamente. Il calcolatore non puo' decidere nulla in autonomia; se il risultato delle sue elaborazioni e' errato la colpa va attribuita unicamente ad un programma mal concepito. Non pensare mai che il computer commetta errori: ricorda che esegue sempre e soltanto cio' che tu gli hai detto di fare.

Come e' possibile allora comunicare col computer? Il problema e' risolto usando uno dei vari linguaggi che via via sono stati concepiti appositamente dagli specialisti del settore. Il piu' usato attualmente nei PERSONAL COMPUTERS e' certamente il linguaggio BASIC. La parola deriva dalla frase inglese ''Beginner's All purpose Symbolic Instruction Code'' (=linguaggio simbolico universale per principianti). Il nome stesso suggerisce che si tratta di un metodo abbastanza semplice per codificare le istruzioni da impartire al computer. Esistono anche altri tipi di linguaggi (FORTRAN, COBOL, PASCAL, per citare solo i piu' noti), ma nessuno e' cosi' semplice e largamente diffuso come il BASIC.

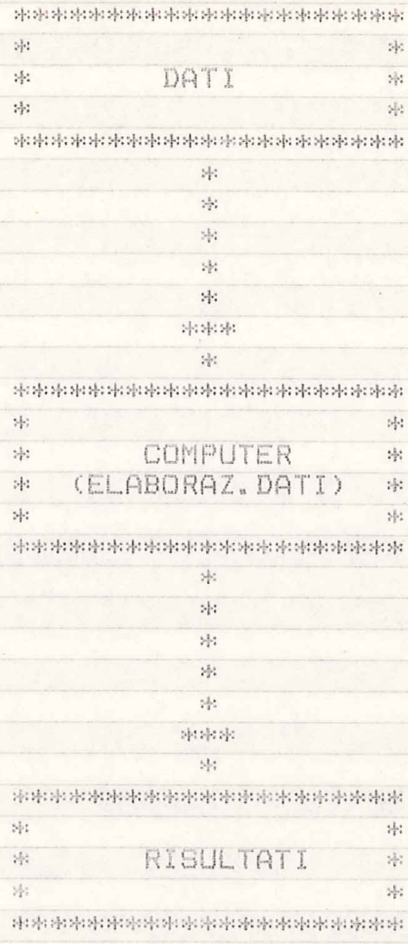
Noi vedremo le istruzioni che si riferiscono al computer Z80 di NUOVA ELETTRONICA, comuni a gran parte dei personal computers in commercio. Il numero di queste istruzioni e' rilevante (circa 170), ma con l'uso frequente diventano familiari e facili da ricordare. Non devi cercare di impararle tutte in fretta: lo faresti male e con grande sforzo. Preoccupati piuttosto di acquisire una buona padronanza delle varie istruzioni, mano a mano che le vedremo.
Non saltare poi alcun argomento: sono tutti piu' o meno collegati tra di loro, e ti troveresti presto in difficolta'. Niente fretta, allora, e ... buon lavoro!

2 - PROGRAMMA, LINEE DI PROGRAMMA, ISTRUZIONI

Un PROGRAMMA BASIC, come abbiamo appena detto, e' un insieme di ISTRUZIONI concepite in modo che i DATI immessi nel computer vengano elaborati per dare i RISULTATI richiesti.

Potremmo anche dire che scopo della programmazione e' quello di indicare alla macchina un certo lavoro da eseguire in modo automatico.

Possiamo schematizzare quanto detto col seguente DIAGRAMMA A BLOCCHI:



Vediamo subito un semplice esempio, in modo da poterci riferire ad esso per chiarire i concetti che sono argomento di questa lezione.

(1)
```
10 REM - SOMMA DI DUE NUMERI
20 A=56
30 B=12.5
40 S=A+B
50 PRINT S
60 END
```

Come vedi, ogni PROGRAMMA BASIC e' composto da una serie di linee numerate in ordine crescente. Il numero della prima linea e' a piacere, come pure l'incremento tra il numero di una linea e quello della linea successiva.
Esistono solo due limitazioni: il computer non accetta numeri di linea minori di zero o maggiori di 65529. Accetta quindi anche il numero di riga zero.

Comunque i programmi sono generalmente scritti partendo dalla linea 10 e proseguendo di dieci in dieci. Cosi' facendo restano disponibili 9 numeri tra una qualsiasi linea e la successiva, e questo torna molto comodo qualora si debbano aggiungere righe.

Tra poco vedremo un'applicazione di questa possibilita', e ti renderai conto che e' molto utile poterla usare in sede di elaborazione di un programma (EDITING).

Esaminiamo il programma partendo dalla prima linea. Troviamo subito una PAROLA CHIAVE del BASIC: REM.

Cosa sono le PAROLE CHIAVE? Sono quelle che il computer e' in grado di riconoscere. Ricorda una cosa molto importante: se usi parole diverse da quelle chiave il sistema ti segnala errore; per la stesura di qualsiasi programma devi percio' adoperare parole scelte unicamente tra quelle ammesse. Vedere a questo proposito l'appendice n.1 a pagina 6 dell'introduzione.

3 - REM - (REMARKS = ANNOTAZIONI)

All'inizio o nel corso di un programma puo' essere molto utile inserire delle annotazioni di commento; in tal modo, quando si va a leggere il LISTATO del programma, quelle note aiutano a capire di che cosa si tratta o cosa avviene in un certo punto.
L'ISTRUZIONE REM E' SEMPRE IGNORATA DAL COMPUTER, che quando la incontra passa a quella successiva (anche se si trova nella stessa linea). E' quindi l'unico caso in cui puoi scrivere parole a tuo piacimento, diverse da quelle chiave.
L'ISTRUZIONE REM PUO' ESSERE ABBREVIATA CON UN APOSTROFO (premi i tasti SHIFT e 7 ed otterrai il simbolo ' che puo' appunto sostituire le lettere REM.)

Nell'esempio 1 la linea 10 serve quindi solamente per ricordarci che quel programma calcola la somma di due numeri.

Proseguiamo con le linee 20, 30 e 40, dove le VARIABILI A, B e S sono poste rispettivamente uguali a 56, 12.5 e alla somma delle due variabili A e B.

Fermiamoci ancora per dire due cose. La prima e' che i numeri decimali, che siamo abituati a scrivere con la virgola, vanno invece introdotti sempre col punto al posto della virgola; in caso contario il programma dara' risultati sbagliati.

La seconda precisazione riguarda uno dei concetti piu' importanti della programmazione in BASIC: quello di VARIABILE. Rifornisciti quindi di pazienza, ossigena bene il cervello e cerca di apprendere meglio che puoi quanto segue!

4 - VARIABILI NUMERICHE -

Uno degli impieghi piu' frequenti del computer e' quello di utilizzarlo per eseguire calcoli matematici. Nel corso di un programma normalmente non si trattano direttamente dei numeri, ma delle variabili che li rappresentano. Questo e' molto pratico perche' tutte le volte che serve ad esempio il numero 12.5 basta scrivere la variabile che ha quel valore (B nel nostro esempio).
Inoltre, e questo e' ancora piu' importante, il valore di una variabile puo' cambiare man mano che il programma prosegue; si intuisce quindi che in casi simili e' assai piu' agevole trattare una variabile, espressa sempre con la stessa lettera, piuttosto che un numero che cambia continuamente.

Vediamo ora come possiamo esprimere le variabili numeriche.

POSSIAMO INDICARE UNA VARIABILE CON UNA QUALSIASI DELLE LETTERE DELL'ALFABETO INGLESE. POSSIAMO ANCHE FARE USO DI DUE CARATTERI, IL PRIMO DEI QUALI DEVE ESSERE COMUNQUE UNA LETTERA, E IL SECONDO PUO' ESSERE UNA LETTERA O UNA CIFRA.
IL SISTEMA ACCETTA E TRATTA ANCHE VARIABILI FORMATE DA PIU' DI DUE CARATTERI: IN TAL CASO PERO' SOLO I PRIMI DUE SONO SIGNIFICATIVI. IL NOME DI UNA VARIABILE NON DEVE CONTENERE PAROLE CHIAVE RISERVATE AL BASIC.

L'insieme dei caratteri che identificano una variabile viene chiamato appunto IDENTIFICATORE.

Ecco qualche esempio di variabile numerica ammessa:

(2) A Y QE LU B3 XX QWERTY LUCA AMICO LU28J53

Da notare che la quarta variabile, l'ottava e la decima corrispondono in effetti ad un unico identificatore: LU. Tra poco vedrai altri esempi che serviranno a chiarirti bene questi concetti.
Gli identificatori di variabile che seguono sono invece illeciti:

(3) 2F R. REMIGIO OTTO STORIA E-G (S)

Infatti eseguendo il primo esempio non si farebbe altro che l'introduzione nella memoria centrale del computer della linea di programma numero 2 (contenente solo la lettera F). Il secondo esempio e gli ultimi due sono illeciti in quanto contengono caratteri non ammessi (punto, trattino, parentesi). I tre casi centrali sono rifiutati dal sistema perche' contengono parole chiave (REM, TO, TO e OR).

Ora, per controllare se hai ben capito, prova a rispondere alle domande che seguono.
A stampante ferma, segui le indicazioni del monitor: le risposte vanno introdotte da tastiera. Nell'eventualita' che fossero errate, il computer te le ripropone; dopo il terzo tentativo, la stampa riprende dando la risposta corretta e la relativa spiegazione. Ricorda di premere sempre il tasto RETURN per introdurre effettivamente i dati digitati.

ESERCIZIO n.1

Quali sono le variabili numeriche errate nell'elenco seguente?

1 - K3

2 - MARMELLATA

3 - 3K

4 - CIFRA

5 - W

6 - D*3

. . . . . .
. . . . . .
. . . . . .
. . . . . .

RISPOSTA

Le variabili inaccettabili sono la 3, la 4 e la 6. Infatti la terza comincia con un numero, la quarta contiene IF e la sesta ha pure essa un carattere non ammesso (*).

NOTA

Avrai notato che in tutti gli esempi di identificatori di variabile numerica le lettere compaiono sempre in maiuscolo. A questo riguardo occorre fare una precisazione importante: IL SISTEMA CONVERTE AUTOMATICAMENTE LE LETTERE MINUSCOLE IN MAIUSCOLE, TRANNE CHE QUANDO SI TRATTA DI SCRITTE TRA VIRGOLETTE. Quindi quando digiti le istruzioni puoi scrivere tutte le parole chiave e le variabili senza premere il tasto SHIFT delle maiuscole: ci pensera' il computer a memorizzarle in maiuscolo. Quando invece stai scrivendo stringhe tra virgolette, sappi che il sistema memorizza le lettere in maiuscolo solo se terrai premuto anche il tasto SHIFT assieme ai vari tasti delle lettere.
SUL MONITOR, IN OGNI CASO, VEDI SEMPRE LETTERE MAIUSCOLE, MA SULLA STAMPANTE VEDRAI CARATTERI MAIUSCOLI O MINUSCOLI A SECONDA DI COME LI HAI INTRODOTTI.

Continuiamo il nostro discorso sulle variabili numeriche, parlando di SINGOLA e DOPPIA PRECISIONE.
In mancanza di particolari istruzioni, il computer esegue i calcoli matematici lavorando con sei cifre significative. Ad esempio, scrivendo alla tastiera:

(4) PRINT 11/7

si otterra' il numero 1.57143 di sei cifre. La sesta cifra risulta arrotondata. Vedremo piu' avanti che esiste anche il modo di eseguire i calcoli matematici lavorando con 16 cifre significative; in tal caso il risultato del calcolo anzidetto sarebbe il numero 1.571428571428571. Come si vede, il risultato precedente e' l'arrotondamento di questo.

Nel primo caso si parla di SINGOLA PRECISIONE del calcolo; nel secondo caso di DOPPIA PRECISIONE.

Il motivo per cui il computer non esegue i calcoli sempre in doppia precisione e' intuitivo: i numeri in doppia precisione occupano piu' memoria di quelli in semplice precisione. Non solo; se i numeri sono considerati interi, occupano ancora meno spazio, sempre ragionando in termini di memoria.

Per tutti questi motivi, ESISTONO TRE TIPI DI VARIABILI NUMERICHE: INTERE, in SINGOLA PRECISIONE, in DOPPIA PRECISIONE.
Se l'identificatore di una variabile e' seguito dal simbolo % la variabile viene trattata come numero intero; se il simbolo e' ! la variabile e' in singola precisione; se il simbolo e' # essa e' in doppia precisione. Ricordare che senza alcun simbolo una variabile numerica e' considerata sempre in singola precisione.
Ecco qualche esempio di variabile numerica ammessa:

```
(5)       E%  KE!  C#  LIMITE%  LIQUORI%  A  A%  A!  A#
```

LIMITE% e LIQUORI% identificano la stessa variabile intera LI, in quanto il computer considera solo i primi due caratteri. Fai bene attenzione agli ultimi quattro esempi: non si tratta di un'unica variabile numerica, ma di tre ben distinte. Infatti le variabili A e A! coincidono, come abbiamo detto poco fa; A% A! A# SONO TRE VARIABILI NUMERICHE COMPLETAMENTE INDIPENDENTI LE UNE DALLE ALTRE. Il computer le tratta separatamente, assegnando ad ognuna di esse il valore che le compete.

Esiste poi ancora un altro tipo di variabile possibile, contraddistinta da un simbolo diverso: si tratta delle variabili di stringa. Esse non contengono numeri ma stringhe, ossia un insieme di caratteri alfanumerici (lettere, numeri ed altri caratteri di qualsiasi tipo).
LE VARIABILI DI STRINGA SONO SEGUITE DAL SIMBOLO $. Vedremo tra poco come vengono definite. Per ora ci serve solo sapere che esistono, che contengono delle diciture di qualsiasi tipo, e che sono contraddistinte dal simbolo $.

Ricapitolando, ogni identificatore di variabile che sia seguito dal simbolo $ rappresenta una stringa e non ha nulla a che fare con una variabile numerica che porti lo stesso nome. Nell'esempio seguente siamo in presenza di quattro variabili diverse ed indipendenti; la prima e' una stringa, le altre sono numeri:

```
(6)       Q$  Q#  Q  Q%
```

E' cosi' terminata la LEZIONE 1.

Il programma, dopo la stampa del riepilogo, proseguira' solo sul monitor. Ti verra' chiesto di digitare alcuni esempi di identificatori di variabili numeriche.
L'argomento e' di estrema importanza per una corretta programmazione in BASIC, quindi cerca di rispondere con criterio. Il computer esamina le tue risposte e ti dice se sono corrette oppure no. Se non ti riesce di darle esatte, rileggi il testo della LEZIONE, poi riprova.
Quando compare la scritta SBAGLIATO, tenendo premuto un tasto qualunque il programma si ferma; in tal modo puoi cercare di scoprire l'errore commesso. Dopo aver lasciato il tasto vedrai scritto il tipo di errore compiuto.
Il computer ti chiedera' per 30 volte di definire una variabile. Alla fine ti dira' quante risposte esatte e sbagliate avrai dato.
Se non vuoi arrivare alla fine dei 30 esercizi, devi battere il numero zero.

RIEPILOGO DELLA LEZIONE 1

Facciamo un breve riassunto degli argomenti trattati in questa prima lezione. Il contenuto delle parentesi accanto alle parole chiave rappresenta l'eventuale abbreviazione della stessa.

PROGRAMMA

Un programma e' un insieme di istruzioni che servono per elaborare dati e fornire i risultati di tale elaborazione.

LINEE DI PROGRAMMA

Le linee di programma sono costituite dalle istruzioni. Possono contenerne anche piu' di una. In tal caso le varie istruzioni sono separate da due punti. Ogni linea deve essere preceduta da un numero progressivo.

PAROLE CHIAVE

Le parole chiave sono le istruzioni ed i simboli che il computer e' in grado di riconoscere. L'uso di parole diverse comporta una segnalazione d'errore.

Es. REM PRINT : . # END

REM (')

La parola chiave REM serve per inserire delle annotazioni. E' ignorata dal computer durante l'esecuzione del programma. Puo' essere abbreviata con il simbolo dell'apostrofo (').

```
Es.     10 REM ***** PROGRAMMA PER RICEVUTA FISCALE *****
```

VARIABILI NUMERICHE

Le variabili numeriche contengono i dati numerici; essi in genere variano durante lo svolgimento del programma. Possono essere indicate con uno o due caratteri il primo dei quali deve essere una lettera, mentre il secondo puo' essere anche un numero. Identificatori di variabili piu' lunghi di due caratteri sono accettati dal sistema; tuttavia solo i primi due sono significativi.
Le variabili numeriche sono di tre tipi: intere (quelle seguite dal simbolo % - sei cifre), in singola precisione (nessun simbolo oppure ! - sei cifre), in doppia precisione (simbolo # - 16 cifre).
La separazione tra parte intera e decimale si deve fare con un punto. Le variabili seguite dal simbolo $ rappresentano delle stringhe.

```
Es.     300 C=38 : G3=B*C+7 : A#=290100.7391337568
```

L E Z I O N E n.2

INDICE

1 - ESERCIZI -

Riprendiamo l'esempio di programma della lezione precedente:

```
(1)     10 REM - SOMMA DI DUE NUMERI
        20 A=56
        30 B=12.5
        40 S=A+B
        50 PRINT S
        60 END
```

Risolvi ora gli esercizi che seguono.

ESERCIZIO n. 1

In quale numero di linea e' definita la variabile B?

.....
.....
.....
.....

RISPOSTA

La variabile B e' definita alla linea numero 30. Nella linea 20 c'e' la definizione della variabile A.

ESERCIZIO n. 2

In quale numero di linea viene eseguita la somma delle due variabili A e B?
.....
.....
.....
.....

RISPOSTA

La somma e' eseguita alla linea 40; il risultato e' posto nella variabile S.

Proseguiamo nell'analisi del programma dell'esempio. Alla linea 40 la variabile S e' definita come la somma di A e B. Alla riga 50 troviamo una nuova parola chiave: PRINT. Essa serve per visualizzare sul monitor il valore della variabile S: infatti la linea 50 significa 'STAMPA S'. Il computer, arrivato a questo punto, ci fara' vedere quindi il valore assunto da quella variabile.

2 - PRINT - (=STAMPA)

Durante lo svolgimento di un programma, quando si arriva all'istruzione PRINT si ottiene sul monitor la stampa di cio' che e' stato richiesto: se PRINT e' seguito da una variabile, si ha il suo valore; se invece dopo PRINT si mette una frase tra virgolette, si ottiene la stampa della frase stessa.
In particolare, se PRINT non e' seguito da nulla, si ottiene il salto di una riga nella stampa sul monitor.

Chiariamo questi concetti modificando un po' il programma del nostro esempio.

(2)
```
10 'SOMMA DI DUE NUMERI
20 A=56
30 B=12.5
35 PRINT A,B
40 S=A+B
45 PRINT
50 PRINT"LA SOMMA DEI NUMERI VALE" ; S
60 END
```

Vediamo con ordine i cambiamenti che sono stati fatti. Nella linea 10 alla parola chiave REM e' stato sostituito il simbolo ' (che serve allo stesso scopo, co-

me e' stato detto nel paragrafo 3 della prima lezione).
La nuova linea 35 fa stampare sul monitor i valori delle variabili A e B.
Come vedi, e' stato facile apportare questa modifica al programma precedente: si e' introdotto un nuovo numero di linea tra il 30 e il 40, grazie alla spaziatura di 10 in 10 che avevamo adottato nella numerazione delle istruzioni.
Analogamente, abbiamo introdotto la linea 45, che ordina al computer di lasciare una riga prima di eseguire la stampa successiva. La linea 50 e' stata cambiata inserendo la frase tra le virgolette.

Avrai notato che nella linea 35 tra A e B compare una virgola, mentre nella 50 tra la fine della frase e S c'e' un punto e virgola.
Questa e' una differenza molto importante ai fini della visualizzazione dei dati nel modo desiderato.
Infatti, mettendo la virgola, il dato successivo viene stampato ad una distanza di 16 caratteri dal precedente; col punto e virgola, invece, la stampa prosegue senza lasciare spazi intermedi.

A questo punto, riguardando la linea 35, potresti pensare che il monitor ti presenti il valore di A, cioe' 56, all'inizio della riga e il valore di B, cioe' 12.5, a cominciare dal sedicesimo carattere della stessa riga. In effetti non e' cosi', perche' LA VISUALIZZAZIONE DI VARIABILI NUMERICHE AVVIENE SEMPRE LASCIANDO UN POSTO ALL'INIZIO PER L'EVENTUALE SEGNO MENO (-), ED UN POSTO LIBERO ALLA FINE DEL NUMERO (questo proprio per non attaccare mai due numeri, anche nel caso che si sia ordinata la loro stampa inserendo tra di essi il punto e virgola, che di per se' non lascia spazi vuoti).

Tornando al caso della linea 35, allora, avremo che nella riga di stampa dei dati, iniziando dalla sinistra, il computer lascera' uno spazio vuoto, poi stampera' il numero 56, poi, ad una distanza di 15 caratteri dalla fine del numero 56, visualizzera' il numero 12.5.

Se tieni conto di quanto detto, in realta' il primo numero e' stampato dal primo carattere della riga (ricorda il segno, che se e' positivo come nel nostro caso viene omesso automaticamente); il secondo numero inizia in effetti al sedicesimo carattere (con le stesse considerazioni sul segno).

Ora dovrebbe essere chiaro anche quello che accade alla linea 50: il valore della variabile S non risulta attaccato alla frase tra virgolette, ma in mezzo c'e' uno spazio libero, quello che compete all'eventuale segno. E' evidente che qualora S assumesse un valore negativo, questo spazio vuoto verrebbe ad essere occupato dal segno meno; sarebbe opportuno, in tal caso, allungare la frase tra virgolette aggiungendo uno spazio alla fine, dopo la parola VALE.

ESERCIZIO n. 3

Supponiamo di modificare le linee 20, 30, 35 dell'esempio (2) in questo modo:

```
20 A=-13
30 B=-1793
35 ? A;B
```

(Ricorda che PRINT puo' essere sostituito dal punto interrogativo!)
Quale delle seguenti soluzioni di visualizzazione e' quella giusta?
(Il simbolo > sta ad indicare il limite sinistro della riga del monitor; fai attenzione agli spazi!)

1 - >-13-1793

2 - > 13-1793

3 - >-13 -1793

4 - > -13 -1793

.....
.....
.....
.....

RISPOSTA

Il monitor visualizza come nel caso 3, cominciando dal primo carattere e lasciando uno spazio tra il primo numero ed il segno meno del secondo. Da notare che se i valori di A e di B fossero stati positivi, ovviamente il segno meno non sarebbe stato stampato, ma i numeri 13 e 1793 sarebbero stati visualizzati sempre nelle posizioni del caso 3.

---

3 - LET - (=SIA)

L'ultima versione del programma di somma di due numeri era la seguente:

(3)

```
10 'SOMMA DI DUE NUMERI
20 A=56
30 B=12.5
35 PRINT A,B
40 S=A+B
45 PRINT
50 PRINT"LA SOMMA DEI NUMERI VALE" ; S
60 END
```

Le linee di programma 20, 30, 40 potrebbero anche essere scritte cosi':

(4)

```
20 LET A=56
30 LET B=12.5
40 LET S=A+B
```

La linea 20, ad esempio, significa 'SIA A=56'.
Il programma girerebbe nello stesso identico modo, perche' per il computer l'indicazione di LET e' facoltativa.
E' stata inserita anche questa possibilita' perche' alcuni computers la richiedono obbligatoriamente: in tal modo allora i programmi fatti su questo calcolatore sono compatibili anche con gli altri. Si tratta comunque di una cosa del

tutto marginale, che puo' essere tranquillamente ignorata, ed e' esattamente quello che faremo d'ora in poi.

4 - END - (=FINE)

Quando il calcolatore arriva a questa istruzione, il programma ha termine. Anche per questa parola chiave valgono considerazioni in parte analoghe a quelle fatte per LET. Infatti molti computers richiedono che ogni programma termini con l'istruzione END, altrimenti segnalano errore. Non e' cosi' nel nostro caso, quindi gli esempi fatti finora sono ugualmente validi e funzionanti anche senza la linea che contiene END.
Avremo pero' occasione di vedere programmi dove l'uso di END e' necessario. Un caso tipico in cui cio' si verifica e' nell'uso delle SUBROUTINES.
D'ora in poi useremo END solo nei casi in cui sia indispensabile.

5 - LINEE CON ISTRUZIONI MULTIPLE -

Fino a qualche anno fa non era possibile mettere piu' di una istruzione per linea. Attualmente invece la maggior parte dei computers accetta anche righe di programmazione con istruzioni multiple, ognuna delle quali e' separata da quella successiva da due punti.
Il limite all'impiego di piu' istruzioni nella stessa linea e' dato dal numero massimo di caratteri in essa inseribili. Normalmente questo numero vale 40 o 80; nel nostro computer ogni linea ha una capienza massima di 255 caratteri, cifra davvero ragguardevole. Tale elevata capienza risulta spesso di grande utilita' pratica, come avremo modo di approfondire.
Per vedere un'applicazione di questa possibilita', dai un'occhiata al seguente programma:


```
(5)     10 REM ** SOMMA DI DUE NUMERI ** : A=56 : B=12.5 : PRINT A,B :
           S=A+B : PRINT : PRINT"LA SOMMA DEI NUMERI VALE"; S : END
```

Si tratta, come avrai notato, dell'esempio 4 scritto in una sola linea! Proviamo, per esercizio, a rendere piu' conciso possibile il programma:

```
(6)     10 'SOMMA DI DUE NUMERI:A=56:B=12.5:?A,B:S=A+B:?:
           ?"LA SOMMA DEI NUMERI VALE";S
```

Abbiamo eliminato tutti gli spazi; REM e' stato sostituito dall'apostrofo; al posto di PRINT abbiamo digitato ?; END e' stato soppresso. Niente male, no?

Facciamo comunque qualche osservazione in proposito:

- meglio non abusare di questa possibilita' piu' del necessario, altrimenti il programma diventa difficile da leggere; inoltre gli eventuali rinvii ad una istruzione che si trova in mezzo ad una riga diventano impossibili.
- tutti gli spazi, nella programmazione in BASIC, sono superflui. Mettendoli si ottiene un programma che e' piu' facilmente leggibile. Eliminandoli si risparmia memoria.
- la sostituzione di PRINT col punto interrogativo vale solo in fase di EDI-

TING (=scrittura) di un programma: nel listato comparira' sempre la parola PRINT.
- l'uso di linee con piu' di una istruzione rende piu' veloce l'esecuzione di un programma.

Come si vede, si tratta spesso di elementi in contrasto tra di loro. Volta per volta si tratta di scegliere la soluzione piu' conveniente.

6 - INPUT - (=FA ENTRARE)

Fino ad ora il nostro esempio non ha molto senso pratico, in quanto i numeri da sommare sono compresi nel programma e si possono cambiare solo variando le linee di programmazione 20 e 30.
Per realizzare invece un programma che sia in grado di sommare due numeri a piacimento, occorre fare uso di una istruzione particolare, che riveste una grande importanza in quanto e' quella che permette al computer di chiedere dati all'operatore. Questi dati non sono quindi piu' fissi perche' scritti nel programma, ma sono variabili perche' scelti dall'utilizzatore.

L'istruzione in oggetto e' INPUT. Vediamone subito una possibile applicazione al nostro caso. Osserva questa nuova versione:

(7)
```
10 'SOMMA DI DUE NUMERI
20 INPUT A
30 INPUT B
35 PRINT A,B
40 S=A+B
45 PRINT
50 PRINT"LA SOMMA DEI NUMERI VALE" ; S
```


A parte l'eliminazione della linea 60 perche' superflua, la vera modifica e' stata apportata alle righe 20 e 30. In esse, anziche' imporre i valori di A e B, compare appunto l'istruzione INPUT. Arrivato alla linea 20, il programma si arresta e sul monitor compare un punto interrogativo: il computer sta eseguendo l'istruzione INPUT A, ossia resta in attesa che l'operatore introduca dalla tastiera un valore numerico; questo valore verra' assegnato alla variabile A.
Se scriviamo 56 e premiamo il tasto RETURN, appare un nuovo punto interrogativo: il programma e' alla linea 30 ed il computer resta in attesa del valore numerico da asegnare alla variabile B. Se introduciamo il valore 12.5, subito dopo vedremo stampati i numeri introdotti (linea 35) e poi la dicitura LA SOMMA DEI NUMERI VALE 68.5.
Il programma e' terminato ed il risultato, ovviamente, e' uguale all'esempio precedente, dal momento che abbiamo introdotto gli stessi valori numerici.

Se ora facciamo ripartire il programma (con RUN, naturalmente), al comparire dei due punti interrogativi possiamo introdurre i numeri che preferiamo: il calcolatore eseguira' la loro somma.
Attenzione, pero': i numeri non devono essere lunghi piu' di sei cifre, altrimenti il risultato potra' essere errato in quanto viene approssimato.

Infatti, se proviamo ad introdurre i numeri 1234567 e 21, il risultato sara' 1.23459E+06. Cosa e' successo? Ricordi il discorso fatto a proposito di variabi-

li numeriche in semplice e doppia precisione? Allora ricorderai anche che se una variabile non porta uno dei simboli # ! % essa viene considerata in singola precisione (esattamente come se fosse seguita dal simbolo !). Cio' significa che il computer assegna alla variabile solo una lunghezza di sei cifre: l'eventuale parte eccedente viene trascurata e serve solo per arrotondare l'ultima delle sei cifre presentate.

L'argomento e' molto importante, quindi vale la pena di approfondirlo ulteriormente. Per poterlo fare in modo molto veloce ed istruttivo, portiamo ancora un cambiamento al nostro programma di somma di due numeri. Per renderlo piu' presentabile e d'uso piu' facile, ci servono altre due istruzioni nuove: CLS e GOTO. Vediamole.

7 - CLS - (CLEAR SCREEN = PULISCI LO SCHERMO)

CLS e' una istruzione facile da capire e da usare: tutte le volte che la incontra, il computer cancella lo schermo del monitor. Le successive istruzioni di stampa avvengono a partire dall'alto.

8 - GOTO - (=VAI A)

L'istruzione GOTO serve per far compiere al programma dei salti di linea: anziche' proseguire alla linea successiva, esso prosegue alla linea specificata. Il salto avviene sempre ed in ogni caso, e non se si verificano certe condizioni; per questo motivo si parla di SALTO INCONDIZIONATO. Tale dicitura e' usata per distinguerlo dal salto che avviene solo al verificarsi di certe condizioni; vedremo in una delle prossime lezioni questo secondo caso.

E' importante cercare di imparare la terminologia normalmente usata in informatica, ossia nel settore specifico dei computers; solo cosi' la lettura di testi e riviste non presentera' piu' incognite. Per questo, mano a mano che andiamo avanti, diamo la spiegazione dei vari termini: la loro esatta comprensione ne facilitera' anche l'apprendimento. Alla fine di questo corso di BASIC in autoistruzione daremo anche un glossario dei termini piu' comuni. In esso le parole d'uso piu' frequente saranno spiegate diffusamente, in modo da poterle consultare velocemente in caso di dubbio.

Vediamo allora la versione aggiornata del programma di somma di due numeri:

(8)

```
10 'SOMMA DI DUE NUMERI
20 CLS
30 INPUT "PRIMO NUMERO" ; A
40 PRINT
50 INPUT "SECONDO NUMERO" ; B
60 PRINT : PRINT A,B : PRINT : PRINT
70 PRINT "LA LORO SOMMA VALE : " ; A+B
80 PRINT : PRINT : PRINT : PRINT : GOTO 30
```

Vedremo nella lezione 3 la spiegazione dettagliata di questo programma.

Dopo la stampa del riepilogo, questa lezione termina con l'esecuzione del programma appena proposto. Potrai cosi' renderti conto del suo funzionamento prima di vederlo spiegato.

Quando il computer ti chiede un numero, introducilo da tastiera facendolo seguire da RETURN; vedrai poi come il computer considera i due numeri introdotti e vedrai la loro somma. Controlla con carta e penna per renderti conto degli errori introdotti dall'approssimazione delle sei cifre.

Fornisci numeri a caso, lunghi a piacere; ad esempio 2902.227453198006 oppure 0.00071339. Il numero ti verra' richiesto qualora tu introduca lettere e non cifre; in tal caso appare la scritta REDO che indica appunto un errore nel fornire dati in INPUT. Se premi i tasti della virgola o dei due punti vedrai la scritta EXTRA IGNORED; significa che le cifre date dopo quei simboli non vengono considerate. La spiegazione di questo comportamento verra' fornita quando vedremo l'istruzione LINEINPUT.

Il programma prosegue all'infinito, a meno che tu non dia zero come risposta al primo numero richiesto; in tal caso la lezione prosegue. Fa quindi molte prove, cercando di prevedere il risultato che ti dara' il computer.

RIEPILOGO DELLA LEZIONE 2

PRINT (?)

L'istruzione PRINT, abbreviabile col punto interrogativo, serve per visualizzare sul monitor i dati specificati. Con una sola istruzione di PRINT si puo' chiedere la stampa di piu' dati; in tal caso essi vanno separati o con una virgola o con un punto e virgola. Nel primo caso si ottiene una spaziatura di 16 caratteri, mentre nel secondo i dati vengono visualizzati uno di seguito all'altro. Per stampare una frase occorre metterla tra virgolette. Gli esempi che seguono danno tra parentesi il risultato della stampa.

```
Es.     PRINT 100,245                          (100             245)
Es.     ? 48.002;16                            (48.002  16)
Es.     PRINT "NUOVA" ; "ELETTRONICA"          (NUOVAELETTRONICA)
Es.     PRINT "NUOVA" , "ELETTRONICA"          (NUOVA           ELETTRONICA)
```

LET

LET serve per assegnare il valore alle variabili. Puo' essere omesso.

Es. 100 LET H3=3.1415

END

L'istruzione END pone fine all'esecuzione del programma. Generalmente si usa quando si hanno delle subroutines; in tal caso il programma non deve arrivare a quelle linee, altrimenti si avrebbe una segnalazione d'errore.

Es. 3190 END

ISTRUZIONI MULTIPLE

Una linea di programma puo' contenere diverse istruzioni, che vanno separate dai due punti. Il limite a questa applicazione e' imposto dal numero massimo di caratteri che una riga puo' contenere: 255.

Es. 230 AF%=13 : C=E*AF% : PRINT AF%

INPUT

Quando il programma arriva ad una istruzione INPUT esso si ferma in attesa di uno o piu' dati che vanno introdotti da tastiera.

Es. 40 INPUT D,E
Es. 95 INPUT "QUANTI ANNI HAI" ; E%

CLS

L'istruzione CLS cancella il monitor.

Es. 430 CLS

GOTO

Con l'uso dell'istruzione GOTO si ottiene un salto incondizionato dalla riga in cui si trova GOTO a quella specificata.

Es. 110 GOTO 65

L E Z I O N E n.3

INDICE

1 - ESEMPIO -

Riprendiamo l'esempio di programma della lezione precedente:

(1)

```
10 'SOMMA DI DUE NUMERI
20 CLS
30 INPUT "PRIMO NUMERO" ; A
40 PRINT
50 INPUT "SECONDO NUMERO" ; B
60 PRINT : PRINT A,B : PRINT : PRINT
70 PRINT "LA LORO SOMMA VALE : " ; A+B
```

```
80 PRINT : PRINT : PRINT : PRINT : GOTO 30
```

Vediamolo in dettaglio. La linea 20 contiene l'istruzione CLS che provoca la cancellazione del monitor. La 30 e' quella che chiede all'operatore di introdurre il primo numero. Se vai alla pagina 7 della lezione 2 a rivedere l'esempio 7, ti puoi rendere conto che ci sono due modi per utilizzare l'istruzione INPUT. Vediamoli riscrivendo la linea 20 dell'esempio 7 e la linea 30 dell'esempio 1:

```
20 INPUT A
30 INPUT "PRIMO NUMERO" ; A
```

In entrambi i casi il computer si ferma ed aspetta un dato numerico. Nella riga 20 la richiesta viene evidenziata con un punto interrogativo, mentre la 30 serve per anteporre la frase PRIMO NUMERO al punto interrogativo. Questo secondo modo e' sicuramente migliore del primo, poiche' la comparsa del solo punto interrogativo non specifica quale sia il dato da introdurre. La persona che usera' il programma non puo' quindi immaginare il tipo di richiesta avanzatagli dal calcolatore: ad esempio, potrebbe pensare di dover introdurre la data oppure il proprio nome. Il secondo metodo elimina invece ogni possibilita' di equivoco.

Il FORMATO di INPUT, ossia il modo in cui va scritta l'istruzione, varia nei due casi, come risulta chiaramente. Sul primo c'e' poco da aggiungere, se non che lo spazio tra INPUT e A puo' essere omesso. Gli spazi possono essere eliminati anche nel secondo caso; dopo la chiusura delle virgolette e' indispensabile mettere il punto e virgola prima dell'identificatore della variabile numerica usata.

Proseguendo nell'esame dell'esempio (1), arriviamo alla linea 40, che serve per saltare una riga nella stampa sul monitor; se la 40 non esistesse, la frase SECONDO NUMERO della linea 50 apparirebbe subito sotto alla precedente scritta PRIMO NUMERO.
Niente di nuovo, poi, alla linea 50; la 60 contiene istruzioni multiple, separate dai due punti. Come abbiamo gia' detto nella lezione scorsa, scrivendo PRINT A,B otteniamo la visualizzazione di A all'inizio della riga e quella di B 16 caratteri piu' a destra. Ricorda le considerazioni sul segno.
Come vedi, la stampa del valore della somma non e' piu' fatta utilizzando la variabile S, ma dicendo al computer di stampare direttamente il valore A+B. Il fatto di passare attraverso l'uso della variabile S risulterebbe preferibile qualora la somma dei due numeri venisse richiamata piu' volte nel proseguimento del programma; in quel caso sarebbe piu' semplice trattare S che non A+B, in

quanto basterebbe una battuta (S) invece delle tre richieste dalla scrittura A+B.
Siamo infine arrivati alla linea 80, anche questa contenente piu' di una istruzione. Le prime quattro (PRINT) servono per saltare 4 righe. L'ultima e' quella che permette la ripetizione all'infinito del programma, in quanto dice al computer di andare alla linea di programma numero 30, in cui viene chiesto nuovamente il primo numero, poi il secondo, e cosi' via.

L'istruzione GOTO riveste pertanto un'importanza notevole in quanto permette di saltare da una parte ad un'altra del programma. Cio' talvolta e' molto comodo, come si vede nell'esempio (1). Bisogna pero' cercare di farne un uso corretto e moderato. Infatti l'impiego sconsiderato di GOTO rende molto difficile la lettu-

ra di un listato di programma, e risulta percio' piu' arduo individuare eventuali errori.

Ad esempio, il programma seguente e' perfettamente equivalente all'ultimo che abbiamo visto. Il listato e' comunque praticamente incomprensibile.

```
10' SOMMA DI DUE NUMERI
20 CLS : GOTO 70
30 PRINT : PRINT A,B : PRINT
40 GOTO 90
50 PRINT : INPUT "SECONDO NUMERO" ; B
60 GOTO 30
70 INPUT "PRIMO NUMERO" ; A
80 GOTO 50
90 PRINT : PRINT : PRINT "LA SOMMA VALE" ; A+B
100 GOTO 120
110 PRINT : PRINT : GOTO 70
120 PRINT : PRINT : GOTO 110
```

Questo esempio e' volutamente esagerato per farti capire che l'istruzione GOTO va usata con criterio.

2 - VARIABILI NUMERICHE IN DOPPIA PRECISIONE -

Eseguendo il programma della somma di due numeri ti sarai reso conto degli errori che il computer introduce coi numeri che superano la lunghezza di sei cifre. Per ovviare a questo inconveniente dobbiamo far ricorso all'uso delle variabili numeriche in doppia precisione. Esse consentono l'uso di 16 cifre, quindi si puo' arrivare al numero 9.999.999.999.999.999 davvero ragguardevole e bastante per ogni tipo di applicazione commerciale o scientifica. Se un numero supera quel limite, esso viene rappresentato in virgola mobile.
Ad esempio il numero in doppia precisione 1234567890123456 7 (17 cifre) diventa 1.2345678901234 57D+16, ossia 1 virgola la parte decimale, il tutto moltiplicato per 10 elevato alla sedicesima. Notare che invece di E+16 compare D+16 per indicare che si tratta di un numero in doppia precisione. La sedicesima cifra viene arrotondata da 6 a 7.

Modifichiamo per l'ultima volta il nostro esempio, introducendo appunto le variabili in doppia precisione. Useremo gli stessi identificatori, seguiti pero'

dal simbolo # per indicare al computer che vogliamo lavorare con 16 cifre.

(2)

```
10 'SOMMA DI DUE NUMERI
20 CLS
30 INPUT "PRIMO NUMERO" ; A#
40 PRINT
50 INPUT "SECONDO NUMERO" ; B#
60 PRINT : PRINT A#,B# : PRINT : PRINT
70 PRINT "LA LORO SOMMA VALE : " ; A#+B#
80 PRINT : PRINT : PRINT : GOTO 30
```

ESERCIZIO n. 1

Qual'e' il numero della linea che contiene una istruzione di salto incondizionato?

.....
.....
.....
.....

RISPOSTA

L'istruzione di salto incondizionato GOTO compare alla linea 80.

ESERCIZIO n. 2

Quali sono le linee che contengono istruzioni multiple?

.....
.....
.....
.....

RISPOSTA

Solo le linee 60 e 80 contengono istruzioni multiple; queste sono separate le une dalle altre da due punti. I due punti della linea 70 sono nel testo tra virgolette e quindi non servono per separare istruzioni multiple.

---

3 - SCRITTURA DI UN PROGRAMMA -

Alla fine di questa lezione proverai a scrivere i primi programmi. Per prima cosa potresti limitarti a ricopiare quelli visti finora, provando poi a farli girare.
In questa lezione troverai tutte quelle nozioni che ti serviranno per scrivere, listare, eseguire, registrare e leggere da disco questi primi esempi di programmi. Le istruzioni che ti servono per poter cominciare non sono di difficile apprendimento; inoltre le daremo in forma semplice e succinta, riservandoci di tornare su ognuna di esse per darne spiegazioni piu' ampie e complete.
Per spiegare le varie istruzioni ci serve un caso pratico; supponiamo allora di voler scrivere l'esempio 2 della pagina precedente.

4 - NEW - (=NOUVO)

Per accingerti a scrivere un nuovo programma, devi dapprima cancellare quello eventualmente gia' presente in memoria. Se tu non lo facessi, le linee che scrivi vanno ad aggiungersi a quelle gia' in memoria: anziche' digitare un nuovo programma, faresti delle aggiunte o delle modifiche a quello gia' presente nel computer.
Cosi', se alla fine di questa lezione decidi di scrivere la prima linea del nostro esempio, in effetti cambieresti il contenuto della linea 10 del programma presente in macchina, ossia di quello che in questo momento ti da' la lezione 3.
La memoria centrale del calcolatore risulta sgombra solo in due casi: all'accensione della macchina oppure se si e' introdotto il comando NEW.
Per cancellare un programma gia' in memoria e' sufficiente scrivere NEW e successivamente premere il tasto RETURN. Vedrai che lo schermo del monitor viene cancellato ed apparira' la scritta:

```
READY
>_
```

READY (=PRONTO) indica che il computer aspetta istruzioni, ossia che non sta eseguendo un programma.
Inizia quindi a digitare la linea 10, scrivendola cosi' come la vedi a pagina 5. Dopo aver scritto tutto il contenuto della riga, premi RETURN. Cosi' facendo il computer la memorizza e puoi subito passare a scrivere la seconda linea del programma. Se commetti qualche errore di battuta puoi cancellarlo facendo uso del tasto in alto a destra nella tastiera, quello col simbolo 'II'. Se non ti riesce di fare la correzione, premi ugualmente RETURN e poi riscrivi la linea interessata dall'inizio. Vedremo piu' avanti i vari modi di correggere linee di programmazione errate.

5 - LIST - (=LISTA)

Dopo aver finito la scrittura del programma, oppure anche in qualsiasi altro momento, se vuoi vederne il contenuto e' sufficiente che tu scriva LIST, premendo poi RETURN. Sul monitor comparira' immediatamente tutto il programma contenuto nella memoria del computer.
Se vuoi listare il contenuto di una sola linea, per es. quella numero 30, devi

scrivere LIST30 e poi premere RETURN. In questo caso (ma non nel precedente) puoi anche abbreviare scrivendo L30, ed otterrai lo stesso risultato.
Se il programma da listare e' molto lungo, evidentemente non riesce a stare in 32 righe di video; in tal caso il listato scorre velocemente e si fermerebbe solo alla fine delle linee di programmazione. Per fermarsi prima si deve fare un BREAK, nel modo che diremo tra poco.

Nella lezione che tratta dell'EDITING troverai altre nozioni al riguardo.
Per ora quelle date ti saranno sufficienti.

6 - LLIST - (=LISTA SU STAMPANTE)

Il comando LLIST e' analogo al precedente, tranne il fatto che il listato viene fornito dalla stampante. Se dopo avere impartito questo comando non succede nulla, significa che la stampante non e' in linea: o e' spenta, o si e' staccato il cavo che la collega al computer, oppure la carta e' finita. Per poter continuare dovrai provvedere nel modo appropriato, oppure fare un BREAK.
L'istruzione LLIST, contrariamente a quanto visto per LIST, non puo' essere abbreviata.

7 - BREAK - (=INTERROMPI)

Il comando di BREAK si ottiene premendo contemporaneamente i due tasti omonimi. Cosi' facendo si interrompe la funzione svolta dal computer in quel momento. Serve per troncare l'esecuzione di un programma, oppure una stampa sul monitor o sulla stampante, e cosi' via.
Anche la pressione contemporanea dei tasti CTRL e A sortisce lo stesso effetto.

Talvolta puo' succedere, per errori di programmazione, che non si riesca a fare un BREAK; in tal caso non resta che premere il tasto del RESET della scheda CPU; in alternativa, si puo' anche spegnere e riaccendere il computer. In entrambi i casi si perde il BASIC ed il programma in memoria. Tale comportamento e' comune a tutti i computers, e si verifica solo per errori di programmazione (oppure se la macchina ha dei problemi di corretto funzionamento).

8 - RUN - (=CORRI)

Dopo aver scritto correttamente il programma, per vederlo funzionare devi farlo partire scrivendo RUN e premendo RETURN.
Per prima cosa si cancella il monitor, poi appare la scritta PRIMO NUMERO. Fino a che non introduci un numero scrivendolo sulla tastiera, il programma e' fermo. Successivamente il computer ti chiede il secondo numero, dopodiche' fornisce il valore dei due numeri introdotti e della loro somma. Il ciclo poi si ripete all'infinito; per terminare occorre fare un BREAK.
Se ottieni una segnalazione d'errore, significa che una linea risulta sbagliata. Riscrivila correttamente e vedrai che tutto si sistemera'.

9 - SAVE - (=SALVA)

A questo punto puoi registrare su disco il programma appena scritto. Per farlo occorre assegnargli un nome a piacimento, lungo non piu' di 8 caratteri. Il nome deve iniziare con una lettera; dal secondo carattere in poi e' facoltativo usare lettere o numeri. E' invece errato digitare caratteri diversi da lettere e numeri. Scegliamo ad esempio il nome SOMMA1; per effettuare la registrazione su disco devi scrivere:

```
(3)       SAVE"SOMMA1"
```

Anche qui vale il discorso gia' fatto a proposito di lettere maiuscole e minu-

scole: puoi scriverle minuscole, in quanto il computer le trasforma da solo in maiuscole. Questo e' l'unico caso in cui tale processo avviene anche per i caratteri compresi tra virgolette.

Una cosa importante: il calcolatore segnala errore se si da' il comando SAVE per registrare su di un disco che abbia il nastro di protezione contro la scrittura.

Dopo aver premuto RETURN, il floppy parte ed il diodo rosso si accende: sta avvenendo la registrazione desiderata, che ha termine quando il floppy e' nuovamente fermo. Ora puoi anche spegnere il computer, se lo desideri: il programma in memoria verra' logicamente perduto, ma esso si trova sul disco e potrai ricaricarlo seguendo le istruzioni che troverai tra poco.

Quello che segue e' un modo di dare il comando SAVE perfettamente equivalente a quello appena visto:

```
(4)       SAVE"SOMMA1
```

E' stato eliminato lo spazio dopo SAVE e non sono state messe le virgolette finali.
Ancora due considerazioni; prima di tutto ricorda che eventuali spazi compresi nel nome vengono interpretati come la fine del nome stesso. Se assegni al programma da registrare il nome SOMMA 1 (con uno spazio prima della cifra 1), l'incisione avviene ugualmente ma il nome assegnato al FILE (=FILA, come sono chiamati i dati incisi su disco) sara' SOMMA.

L'ultima cosa e' molto importante: se sul disco esiste gia' un file con quel nome, il programma da registrare prende il posto del vecchio cancellandolo. Prima di effettuare una registrazione occorre quindi sincerarsi sempre che il nome che si intende usare non sia gia' compreso nel dischetto. Il comando DIR serve proprio a questo scopo.

10 - DIR - (DIRECTORY = INDICE)

Il comando DIR serve per conoscere i nomi dei files contenuti in un dischetto, e puo' fornire anche altre informazioni che vedremo piu' avanti. Per ora daremo solo le nozioni indispensabili, al fine di non appesantire troppo l'apprendimento di queste prime lezioni. Non ci stancheremo mai di ribadire che e' molto piu' importante la qualita' del lavoro svolto, piuttosto che la sua quantita'. Meglio

procedere con calma, cercando di capire bene l'uso delle varie istruzioni che via via incontreremo. Cerca poi di eseguire tutti gli esercizi proposti e di farne altri simili per conto tuo: solo la pratica personale e diretta trasforma la teoria in bagaglio d'esperienza.

Il FORMATO di DIR, ossia il modo in cui va impartito, e' il seguente:

```
(5)       CMD"DIR :n"
```

Al solito, le virgolette finali si possono omettere; 'n' e' un numero compreso tra zero e 3, ed indica il drive su cui si vuole applicare il comando DIR. Se si vuole il contenuto del disco inserito nel primo drive, il numero zero puo' esse-

re omesso, assieme ai due punti. Occorrre prestare molta attenzione allo spazio che segue DIR; se 'n' viene specificato, quello spazio deve esserci e deve essere di un solo carattere. In caso contrario si ottiene sempre l'indice del primo drive, anche se 'n' non vale zero. Non mettere mai uno spazio dopo i due punti: e' uno dei casi che possono far perdere il controllo del computer e costringere a fare un RESET, come abbiamo detto poco fa, con perdita del BASIC e del programma presente in memoria.

Tieni presente che nel drive zero deve essere sempre montato il disco contenente il BASIC, altrimenti il computer non puo' svolgere le sue operazioni.

Se i files registrati su un dischetto sono molti, puo' succedere che il loro elenco completo non entri in una sola pagina video; in questo caso il monitor fa vedere un elenco solo parziale. Per completarlo occorre premere un tasto qualunque: appariranno i nomi dei rimanenti files, con in fondo la scritta READY.

11 - LOAD - (=CARICA)

Per provare se la registrazione e' andata a buon fine, senza spegnere il computer, puoi fare nel modo seguente. Digita NEW e poi premi RETURN (ricorda che qualsiasi istruzione diventa operante solo dopo aver premuto RETURN; d'ora in poi non lo diremo piu' per non appesantire inutilmente le spiegazioni).
Cosi' facendo nella memoria del computer non esiste piu' alcun programma. Per sincerarsene e' sufficiente introdurre il comando LIST prima descritto: non appariranno linee di programma.

Successivamente puoi dare il seguente comando:

(6) LOAD"SOMMA1"

Il floppy parte, e quando riappare la scritta READY il programma risulta presente nella memoria centrale del calcolatore. Una LIST oppure un RUN lo dimostreranno inequivocabilmente.

Anche per questo comando vale quanto detto per SAVE: virgolette finali e spazi possono essere omessi.

ESERCIZIO n. 1

Qual'e' il modo piu' conciso di impartire il comando di listare la linea 120 di un programma?
.....
.....
.....
.....

RISPOSTA

Per richiedere il contenuto della linea 120 col minor numero possibile di caratteri, devi digitare L120 e poi premere RETURN.

ESERCIZIO n. 2

Qual'e' il modo piu' conciso di impartire il comando DIR per un disco inserito nel drive 1?

.....
.....
.....
.....

RISPOSTA

Il modo piu' breve per chiedere l'indice dei files di un disco inserito nel drive 1 e': CMD"DIR :1.

La lezione 3 e' terminata. Dopo la stampa del riepilogo esegui le prove pratiche descritte alle pagine 6-7-8-9.

RIEPILOGO DELLA LEZIONE 3

VARIABILI NUMERICHE IN DOPPIA PRECISIONE

Le variabili numeriche in doppia precisione contengono fino a 16 cifre, e sono tali quando l'identificatore e' seguito dal simbolo #.

Es. 210 DF#=8008256134.0032
Es. 300 H##=62#+WW-72

NEW

Il comando NEW serve per cancellare un programma presente in memoria.

LIST (L)

Col comando LIST si ottiene il contenuto delle linee del programma presente in memoria centrale. Se si specifica il numero della linea desiderata esso puo' essere abbreviato con L.

Es. LIST LIST200 L200 L10- L. L20-50

LLIST

Il comando LLIST esplica le stesse funzioni di LIST, fornendo pero' i risultati sulla stampante. Vale la casistica scritta per LIST.

BREAK

Col comando BREAK si ottiene l'interruzione delle funzioni svolte dal computer; la pressione contemporanea dei tasti CTRL e A fornisce lo stesso risultato.

RUN

Questo comando fa partire il programma presente in memoria.

SAVE

Questo comando serve per registrare su disco il programma presente in memoria. Il disco non deve avere la protezione contro la scrittura.

Es. SAVE "FATTURE"
Es. SAVE"ELENCO

DIR

Col comando DIR e' possibile conoscere i nomi dei files contenuti in un disco.

Es. CMD " DIR :2"
Es. CMD"DIR

LOAD

Con questo comando si ordina al computer di cercare sul disco il programma che ha il nome specificato e di trasferirlo in memoria. L'eventuale programma prece-

dentemente presente in macchina viene perduto.

Es. LOAD "MOVIM26"
Es. LOAD"MOVIM26

L E Z I O N E n.4

INDICE

1 - BASIC E DOS -

Per poter proseguire agevolmente nella spiegazione del funzionamento del computer, bisogna che analizziamo il suo funzionamento, naturalmente a grandi linee. Il BASIC e' il linguaggio che ci permette di comunicare col calcolatore, come sai; esso pero' non e' contenuto permanentemente nella sua memoria, ma vi viene caricato prelevandolo da un dischetto.

L'esecuzione di tutte le operazioni compiute sui dischi (e ne vedremo parecchie) necessita di molte istruzioni: ad esempio, quando diciamo al calcolatore di leggere un programma registrato sul disco, esso deve fare partire i floppy, poi andarsi a cercare nell'indice del primo disco il file che porta il nome che abbiamo indicato, e se lo trova deve portarsi sulla traccia dove inizia il programma, leggerselo e trasferirlo in memoria; se il programma non si trova sul primo disco, va a cercarlo nel secondo, e cosi' via. Se non trova il file, ci fa una comunicazione in questo senso. Ebbene, tutte queste operazioni non hanno niente a che fare col BASIC, ma sono contenute nel DOS, che e' quindi l'insieme delle istruzioni necessarie al computer per gestire tutte le operazioni che riguardano i dischi. La dicitura DOS deriva dalle iniziali di DISK OPERATIVE SYSTEM: sistema operativo del disco.

Per capire bene il funzionamento del computer, analizziamo il suo comportamento. Quando lo accendi, sul monitor compare la scritta che hai gia' visto tante volte. In pratica, l'unica cosa che puoi fare e' quella di premere un tasto: il primo floppy parte e comincia a scambiare dati col calcolatore; dopo pochi secondi appare la scritta:

```
(1)        READY
           >.
```

In quel breve lasso di tempo e' accaduto all'incirca quanto segue: appena acceso, il computer e' in grado di leggere unicamente il disco contenente il DOS ed il BASIC. All'interno del calcolatore c'e' un particolare circuito integrato, di nome EPROM, che gli ordina di leggersi un ben determinato file sul primo floppy. Inizia cosi' il caricamento del DOS e si prosegue con quello del BASIC. Non e' fuori posto far notare l'enorme velocita' con cui il computer carica i dati; infatti in quei pochi secondi circa 35.000 caratteri passano dal disco alla memoria centrale, ad una velocita' media di trasmissione di 2300 caratteri al secondo; in effetti essa risulta anche piu' grande, pensando che gran parte di quel tempo viene impiegata per cercare i files da caricare.

2 - LIVELLI OPERATIVI -

Quando appare la scritta (1), il computer ci comunica che e' pronto ad eseguire nostri ordini (READY = pronto). Il simbolo > seguito dal cursore invita all'introduzione di istruzioni. Giunti a questo punto, possiamo percorrere due strade: operare in BASIC oppure in DOS; ovviamente la seconda scelta viene effettuata per fare certe operazioni coi dischi che non sono accessibili dal BASIC, come avremo modo di vedere tra poco. Per questo motivo si parla di LIVELLI OPERATIVI; essi sono due e corrispondono al LIVELLO DOS e al LIVELLO BASIC.
Per cercare di spiegare il concetto ora esposto con parole molto semplici, e'

come se abitassimo in una casa a due piani, con la parte giorno al pian terreno e quella notte al primo piano: ad ogni zona corrispondono le funzioni sue pro-

prie. Cosi', non ci sogneremmo mai di dormire sul tavolo di cucina o di mangiare nella toilette. Lo stesso vale per il computer ai due diversi livelli citati, in quanto ognuno di essi e' stato concepito per svolgere compiti ben precisi. Col DOS si lavora sui dischetti e col BASIC si comunica col sistema. DOS e BASIC sono poi interattivi, nel senso che operano in collegamento tra di loro quando ci si trova al livello BASIC: da questo devono essere eseguibili tutte le operazioni sui dischi.

Al livello BASIC e' poi possibile non solo scrivere od eseguire programmi, ma anche effettuare calcoli diretti, usando il computer come se fosse una calcolatrice, oppure scrivere direttamente sulla stampante, come se si trattase di una macchina per scrivere, e cosi' via. Molte istruzioni BASIC sono eseguite anche se vengono impartite fuori da un programma; in tal caso si parla di COMANDI DIRETTI.

Anche se impropriamente, d'ora in poi parleremo di tre livelli operativi: quello DOS, quello BASIC e quello dei comandi diretti (quest'ultimo sarebbe compreso nel secondo).

3 - COMANDI DIRETTI -

Parliamo innanzitutto di alcune possibilita' d'uso del computer a livello di comandi diretti, cominciando dall'esecuzione di calcoli matematici.
Se vogliamo calcolare, ad esempio, il prodotto di 38.03 per 107.2 dobbiamo scrivere direttamente alla tastiera:

```
(2)        >? 38.03 * 107.2
```

Il simbolo > sta a significare che operiamo con comandi diretti. Il punto interrogativo e' l'abbreviazione di PRINT; ricorderai anche che nei numeri decimali si deve fare uso del punto e non della virgola. Inoltre tutti gli spazi non sono indispensabili, e li mettiamo sempre per rendere piu' facilmente leggibile il testo.
Tornando all'esempio (2), dopo aver premuto il tasto RETURN otterremo immediatamente il risultato dell'operazione richiesta, ossia il numero 4076.82, formato dalle solite sei cifre. Chiedendo calcoli nel modo visto or ora, si hanno risultati che sono sempre calcolati in singola precisione, con tutto quello che si e' detto per la rappresentazione eventuale in virgola mobile.

Vediamo un altro esempio, piu' significativo:

```
(3)        >? 29/13
```

Il risultato fornito e' 2.23077; quello esatto sarebbe 2.230769230769. Le prime sei cifre decimali costituiscono il periodo, in quanto si ripetono sempre con la stessa sequenza, senza avere mai fine. Come vedi, la sesta cifra e' approssimata. Per ottenere risultati esatti bisogna lavorare in doppia precisione:

```
(4)        >A#=29 : B#=13 :  ? A#/B#
```


Siamo passati attraverso la definizione di due variabili A# e B# in doppia precisione. cosi' il loro quoziente viene fornito con 16 cifre: il risultato vi-

sualizzato e' infatti 2.2307692307692 31. L'ultima cifra e' arrotondata, come al solito.

Occorre fare attenzione, invece, agli esponenziali; in pratica essi sono calcolati sempre in singola precisione, anche lavorando con 16 cifre: di quelle, solo le prime 6 sono sicure. Prova, ad esempio, a fare il calcolo seguente, impostato con variabili in doppia precisione (^ e' il simbolo che si deve usare per eseguire gli elevamenti a potenza):

(5) >A#=1234567890123456 : B#=1/2 : ? A#^B#

L'operazione chiesta e' la radice quadrata di A#, ed il risultato fornito dal computer e' 3.51364E+07, ossia 35136400. Si vede chiaramente che, nel caso di potenze, il risultato viene dato in singola precisione anche partendo da variabili in doppia precisione.

Potremmo pensare di aggirare l'ostacolo nel seguente modo:

(6) >A#=1234567890123456 : B#=1/2 : C#=A#^B# : ? C#

Il risultato sara' 35136368, che e' un numero in doppia precisione e puo' sembrare esatto. Prova pero' a fare:

(7) >D#=2 : E#=C#^D# : ? E#

In questo modo prendiamo il valore della precedente operazione e lo eleviamo al quadrato: dovremmo ottenere il numero di partenza, quello assegnato ad A#. Invece si ha 1234562445213696, ben lontano da quel valore. Solo le prime sei cifre sono esatte, mentre le rimanenti risultano fasulle.
Tale comportamento con gli esponenziali e' comune a tutti i computers; anzi, il computer Z80 di NUOVA ELETTRONICA e' uno di quelli che risentono meno di questo difetto, come si puo' appurare usando altri calcolatori anche famosi.

4 - CALCOLI MATEMATICI -

Gia' che siamo in argomento, continuiamo a parlare dei calcoli matematici; si tratta di concetti basilari, in quanto ogni programma comporta, di solito, l'esecuzione di molte operazioni di ogni tipo. Su tale argomento le cose da dire sono parecchie, e le vedremo poco a poco, quando se ne presenta l'occasione.

I simboli da usare per eseguire le operazioni sono i seguenti:

(8)

+	per le addizioni
-	per le sottrazioni
*	per i prodotti
/	per le divisioni
^	per gli elevamenti a potenza

Non e' possibile sottintendere il segno *, come spesso si fa in matematica: per

ottenere il prodotto del numero tre per la variabile G non si puo' scrivere 3G, ma si deve fare 3*G.

La divisione si ottiene col simbolo della barra trasversale; il tasto e' quello in basso a destra e non va confuso coi due tasti FS e § che gli stanno sopra e che recano incisa anche una barra trasversale.

ESERCIZIO n. 1

Quale operazione, tra quelle sotto elencate, e' data in modo errato?

1 - PRINT A-3.09

2 - ? R%-16*(T!)

3 - F#=K3#^W+1.1

4 - X=12:3

5 - ZZ=QQ-.13+(AS#*2)

6 - LPRINT 12^(1/6)

.....
.....
.....
.....

RISPOSTA

L'operazione illecita e' la quarta; infatti la divisione si fa col simbolo / e non con i due punti; essi servono solo per separare istruzioni multiple le une dalle altre.

ESERCIZIO n. 2

Quale delle soluzioni seguenti serve per chiedere nel modo migliore il risultato del prodotto di 78 per il numero decimale -0.025?

1 - 78*-.025

2 - 78*(-0,025)

3 - -78*(25/100)

4 - 78*(-1)*25/1000

5 - 78 (-0.025)

RISPOSTA

La risposta migliore e' la prima: col computer e' infatti permesso far seguire il segno - a quello del prodotto, anche senza l'uso della parentesi. Inoltre i numeri decimali con parte intera uguale a zero possono essere scritti trascurando la cifra 0 e scrivendo dal punto in poi.
Delle altre soluzioni, solo la 4 e' corretta, pur se inutilmente laboriosa. La seconda ha la virgola al posto del punto; la terza sarebbe giusta se ci fosse 1000 al posto di 100. L'ultimo esempio e' errato perche' manca il simbolo *.

5 - PRECEDENZA NEI CALCOLI -

Quello della precedenza che il computer assegna ai calcoli e' un argomento molto importante; se non si conoscono queste regole si corre il rischio di commettere errori di calcolo, anche molto gravi. Diamo subito l'elenco delle priorita' seguite dal computer:

1- LE PRIME OPERAZIONI AD ESSERE ESEGUITE SONO QUELLE ENTRO PARENTESI; SE ESISTONO DIVERSI LIVELLI DI PARENTESI, SONO TRATTATE PER PRIME QUELLE PIU' INTERNE. Fuori o dentro le parentesi valgono poi le seguenti priorita':
2- ESPONENZIALI
3- SEGNO MENO
4- MOLTIPLICAZIONI E DIVISIONI
5- ADDIZIONI E SOTTRAZIONI
6- OPERATORI DI CONFRONTO (ossia > < <> >= <= =).

Possono anche sembrare cose ovvie, ma spesso i risultati di un programma sono sbagliati proprio per non aver tenuto conto di queste regole.

Vediamo qualche esempio. Per calcolare la media aritmetica di tre numeri (cioe' la loro somma divisa per tre), si deve scrivere:

```
(9)       (A+B+C)/3
```

Se si scrivesse:

```
(10)      A+B+C/3
```

solo il terzo numero verrebbe diviso per tre, in quanto in mancanza di parentesi la prima operazione eseguita e' la divisione.

Osserva la seguente operazione:

(11) 16^1/2

Non si tratta, come potrebbe sembrare, della radice quadrata di 16, ossia 4, ma di 16 elevato alla potenza 1 ed il risultato viene diviso per 2: si ottiene 8.

Ancora qualche osservazione: il numero delle parentesi aperte deve essere uguale a quelle chiuse; un numero diviso per zero e' errore (codice 11). A questo proposito c'e' da notare che, in presenza di esponenziali, non sempre i risultati sono quelli che ci si aspetta. Osserva il seguente calcolo:

(12) 7-7^(3/3)

Il risultato fornito dal computer e' -1.43051E-06, ossia -0,00000143051, mentre la risposta esatta e' zero. Provando invece a fare 8-7^(3/3) si ottiene il numero 0.999999, e l'operazione 7^(3/3) fornisce il valore giusto, che e' 7.
Sempre per eseguire delle prove con quei valori, il calcolo 6-7^(3/3) da' il numero -1, che va bene.

Con le potenze occorre quindi prestare molta attenzione, studiando ogni volta il modo migliore per affrontare il calcolo.

Niente di meglio, per fugare i dubbi, che fare qualche esercizio.

ESERCIZIO n. 3

Quali sono i risultati di questi quattro calcoli?

12/2^2+2

12/(2^2+2)

(12/2^2)+2

(12/2)^2+2

.....
.....
.....
.....

RISPOSTA

Il risultato della prima operazione e' 5, perche' prima viene effettuata la potenza 2^2 (che da' 4), poi viene calcolata la divisione di 12 per 4 (3), e per ultimo viene sommato 2.
Il secondo calcolo ha per risultato 2: infatti viene prima calcolato il contenuto della parentesi (che e' 6), e successivamente e' eseguita la divisione.
Il terzo conto equivale in tutto e per tutto al primo.
Nella quarta operazione viene calcolata per prima la parentesi (6), poi c'e'

l'elevamento alla potenza 2 (36), quindi si somma 2, ottenendo cosi' come risultato finale il numero 38.

ESERCIZIO n. 4

Se la variabile A vale 3 e B assume il valore 5, quanto vale l'espressione seguente?

A/B/A*B+1

.....
.....
.....
.....

RISPOSTA

Le varie operazioni vengono eseguite nell'ordine in cui sono scritte: prima A/B, ossia 3/5, poi si divide per A, cioe' si ottiene 1/5; successivamente si moltiplica per B, e si ha 1; aggiungendo 1 si arriva al risultato finale 2.
Da notare che l'espressione data assume sempre il valore 2, indipendentemente dai valori assegnati alle variabili A e B (zero escluso).

6 - LIVELLO DOS -

Torniamo al punto in cui eravamo rimasti, per parlare del livello DOS. Il DOS, come abbiamo detto, serve per la gestione di tutte le operazioni che riguardano i dischetti, ed e' un qualcosa di completamente separato dal BASIC.
A livello DOS si possono eseguire diverse operazioni, ognuna delle quali e' messa in atto scrivendo sulla tastiera il relativo comando. Ecco l'elenco completo dei comandi DOS:

(13)

1 - APPEND	11 - FREE
2 - ATTRIB	12 - KILL
3 - AUTO	13 - LIB
4 - BASIC	14 - LIST
5 - COPY	15 - LOAD
6 - DATE	16 - PRINT
7 - DEBUG	17 - PROT
8 - DIR	18 - RENAME
9 - DUMP	19 - TIME
10 - FORMAT	20 - VERIFY

Di questi comandi hai gia' visto un'applicazione di DIR a pag. 8 della lezione 3, dove hai imparato ad usarlo partendo dal livello BASIC.

Se ricordi, dal BASIC si deve scrivere CMD"DIR" per avere l'indice dei files contenuti nel disco che si trova inserito nel drive zero. Ebbene, quello e' pro-

prio il modo per richiedere l'esecuzione di qualsiasi comando del DOS partendo dal livello BASIC (CMD e' l'abbreviazione della parola inglese COMMAND = comando). Se invece ci si trova al livello DOS, tali comandi vanno impartiti senza fare uso di CMD.

Vediamo prima di tutto come si passa dal livello BASIC al livello DOS, e viceversa. Quando accendi il computer, invece di premere la barra dello spazio per effettuare il caricamento del DOS e del BASIC, prova a premere il tasto RETURN e tienilo premuto fino a quando il drive non si ferma; a questo punto lascia il tasto: sul monitor appare la scritta

(14) DOS READY

Tale dicitura indica che sei a livello DOS; non puoi scrivere od eseguire programmi, ma solo dare uno dei comandi DOS.

Per passare dal DOS al BASIC devi ora scrivere sulla tastiera:

(15) BASIC

e premere RETURN. Il floppy parte e dopo circa 7 secondi vedrai la scritta:

(16) READY
>.

Ora sei a livello BASIC. (Per inciso, dato che il BASIC occupa piu' o meno 26000 bytes di memoria, si vede che la velocita' reale di trasmissione dati tra disco e memoria centrale e' di circa 4000 caratteri al secondo).

Quello appena descritto non e' l'unico modo di portarsi al livello DOS; infatti anche dal BASIC si puo' passare al DOS scrivendo:

(17) CMD"S"

Dopo un paio di secondi appare la dicitura (14), la quale indica che ti sei portato al livello DOS.
Come vedi, e' facile passare da un livello all'altro; devi pero' ricordare una cosa molto importante: IMPARTENDO IL COMANDO DIRETTO CMD"S" SI PASSA AL LIVELLO DOS, MA SI PERDE IL BASIC E L'EVENTUALE PROGRAMMA PRESENTE IN MEMORIA CENTRALE. Prima di impartire il comando occorre percio' effettuare il salvataggio di tale programma, se ci interessa conservarlo.
Comunque tale procedura e' indispensabile solo in un caso, come vedremo a suo tempo; in tutti gli altri e' possibile eseguire comandi DOS partendo dal BASIC, con l'uso di CMD.
Se ci si trova a livello DOS, per avere il contenuto del disco inserito nel primo drive basta scrivere:

(18) DIR

e si ha quanto richiesto; come vedi, non si deve fare uso di CMD.

Ricapitoliamo: IL LIVELLO DOS SERVE PER LA GESTIONE DEI DISCHI ED IL LIVELLO BASIC PER SCRIVERE OD ESEGUIRE PROGRAMMI O COMANDI DIRETTI. QUANDO CI SI TROVA AL

LIVELLO DOS SI POSSONO ESEGUIRE DIRETTAMENTE TUTTI I COMANDI DOS DELLA TABELLA (13); ESSI SONO ACCESSIBILI ANCHE DAL BASIC, PERO' IN TAL CASO OCCORRE FARE USO DEL COMANDO DIRETTO CMD"funzione DOS", PONENDO IL COMANDO DOS TRA LE VIRGOLETTE. IN QUESTO SECONDO CASO, DOPO CHE IL COMANDO E' STATO ESEGUITO, IL COMPUTER TORNA AUTOMATICAMENTE AL LIVELLO BASIC, CONSERVANDO IL PROGRAMMA CHE ERA IN MEMORIA.

C'e' da osservare che, qualora il programma lasci disponibili poche celle di memoria, ogni comando CMD viene rifiutato con la scritta:

(19) OUT OF MEMORY IN 15359

Cio' avviene perche' il computer ha bisogno di una certa quantita' di memoria per caricarvi il programma corrispondente al comando DOS richiesto.
Se il numero di celle disponibili e' troppo basso, il calcolatore non puo' pertanto eseguire l'istruzione che gli e' stata impartita, e da' la segnalazione di errore anzidetta. Non resta altro, in questi casi, che salvare il programma su disco col comando diretto SAVE, e poi assegnare il comando NEW; a questo punto, avendo liberato completamente la memoria centrale, il comando CMD viene accettato. Dopo la sua esecuzione, si puo' recuperare il programma con il comando LOAD.

La lezione e' terminata. In quella seguente vedremo l'uso di alcuni comandi DOS che e' necessario imparare subito: COPY, FORMAT, FREE, KILL, LIB, RENAME.
Assieme a quelli gia' visti (BASIC, DIR), formano il bagaglio minimo per poter proseguire.
Quando la lezione e' finita, devi esercitarti a passare dal livello DOS a quello BASIC e viceversa, per familiarizzarti con questi importanti concetti. Prova poi a caricare il DOS senza caricare anche il BASIC, con la procedura descritta poco fa. Esegui infine alcuni calcoli in via diretta, esercitandoti sulle cose dette all'inizio della lezione: una conoscenza approfondita del comportamento del computer nei calcoli matematici e' assolutamente indispensabile per evitare errori in futuro.
L'esecuzione degli esercizi che ti verranno proposti dal computer alla fine del riepilogo dovrebbe aiutarti molto a comprendere la precedenza nei calcoli; gli esercizi sono sempre diversi, in quanto i valori numerici vengono assegnati a caso dal computer. Potrai quindi ripeterli all'infinito, senza dover mai affrontare un calcolo gia' fatto. Per terminare l'esecuzione degli esercizi devi scrivere FINE e premere RETURN.

RIEPILOGO DELLA LEZIONE 4

DOS

Il DOS e' il sistema operativo dei dischi, ossia l'insieme di tutte le istruzioni necessarie per registrare, leggere e gestire tutte le altre operazioni che vanno fatte sui dischetti. Il computer e' in grado di fare tutto questo solo dopo aver caricato il DOS, che si trova inciso sul disco sistema NE-DOS sotto forma di files.

LIVELLI OPERATIVI

Il computer puo' essere usato a due livelli operativi: DOS e BASIC. Del primo si e' gia' detto; il secondo serve per elaborare programmi o per eseguire comandi diretti.

COMANDI DIRETTI

A livello BASIC e' possibile dare anche comandi diretti; l'esecuzione di calcoli alla tastiera, i comandi SAVE, LIST, LOAD, NEW, ecc. ne sono degli esempi.

PRECEDENZA NEI CALCOLI

I calcoli matematici seguono un ordine di precedenza: prima il contenuto delle parentesi, poi gli elevamenti a potenza, quindi il cambiamento di segno, le moltiplicazioni e le divisioni, le somme e le sottrazioni, ed infine gli operatori di confronto.

L E Z I O N E n.5

INDICE

1 - FORMATTAZIONE DEI DISCHETTI -

La formattazione dei dischi e' molto importante, poiche' senza di essa non sarebbe possibile utilizzarli. Vediamo di esaminare in modo approfondito il problema.

Il FLOPPY DISK (= disco morbido) e' formato da una custodia esterna, relativamente rigida, che fa da contenitore per il dischetto vero e proprio. Questo e' costituito da un materiale plastico di supporto, rivestito, su entrambe le facciate, da una sostanza magnetica, piu' o meno come nei comuni nastri magnetici per registratori audio. Il meccanismo entro cui va inserito il disco, chiamato DRIVE (= guida), lo mette in rotazione; un'apposita testina megnetica assolve poi alla funzione di registrare o leggere i dati che vengono scambiati col computer. Tali dati sono sempre sotto forma binaria, ossia sono costituiti da un insieme di impulsi che possono assumere solo due livelli, 1 e 0. Le combinazioni di questi segnali formano, in un certo codice, i vari caratteri alfanumerici, ossia le lettere, i numeri e tutti gli altri simboli grafici che compaiono sulla tastiera. Ci sono poi altri caratteri speciali coi quali il calcolatore rappresenta le parole chiave del BASIC.

Ebbene, quando si effettuano delle registrazioni sul dischetto, si ha un passaggio di impulsi binari, detti comunemente BIT (dalla dicitura inglese BINARY DIGIT = cifra binaria), dalla memoria centrale al disco. Questi dati, per mezzo della testina magnetica, vengono incisi sul disco, dal quale possono poi essere riletti.

Le cose pero' non sono cosi' semplici; infatti un disco vergine, per poter ricevere dati, deve essere prima FORMATTATO (si dice anche INIZIALIZZATO). Cosa significa? Semplicemente che la facciata vergine del disco viene suddivisa, con impulsi elettrici, in 40 circonferenze concentriche (TRACCE), ognuna delle quali e' poi ulteriormente frazionata in 10 archi uguali (SETTORI).
In tal modo si forma una struttura portante dei dati, che possono essere registrati ordinatamente, riempiendo poco a poco i settori e le tracce disponibili. Le 40 tracce concentriche sono numerate dall'esterno verso l'interno, a cominciare dal numero 0 ed assegnando quindi il numero 39 alla pista piu' interna. I 10 settori di ogni traccia sono numerati da 0 a 9, e la loro posizione fa riferimento al piccolo foro che si trova vicino al centro del disco: una cellula fotoelettrica e' in grado di riconoscere il foro e di dare quindi un riferimento per la posizione dei settori.

Non tutte le piste sono disponibili per i dati; la diciassettesima viene normalmente riservata alla DIRECTORY, ossia all'indice del contenuto del dischetto. Abbiamo gia' detto, infatti, che ogni file viene registrato assegnandogli un no-

me; tale nome viene inciso nella directory, che contiene anche le indicazioni per individuare il settore e la traccia in cui quel file inizia.

A questo punto, dovrebbe essere chiaro il concetto di formattazione di un disco vergine: senza di essa i dati non possono venir registrati con un ordine prestabilito; sarebbe come se scrivessimo alla rinfusa tanti caratteri in un enorme foglio bianco, senza righe o quadretti e senza seguire uno schema logico; riusciremmo a sporcare il foglio, ma non a creare una struttura in grado di restituirci una lettera o un romanzo. Solo organizzando la superficie a disposizione

con linee, numeri di pagina, indice, ecc. possiamo dire di aver fatto un libro.

2 - FORMAT - (FORMATTA)

Vediamo ora come si fa a formattare i dischetti vergini. Supponiamo di avere due drive, che e' la configurazione standard per chi ha i floppy disk. La procedura che descriveremo tra poco e' praticamente la stessa anche usando un solo drive o piu' di uno, come vedremo.
Il comando FORMAT appartiene al DOS, quindi e' accessibile sia dal livello DOS che da quello BASIC. Se ci si trova in DOS, ossia se sul monitor compare la dicitura DOS READY, e' sufficiente scrivere alla tastiera:

(1) FORMAT

per avviare il processo della formattazione. Come gia' sai, se invece ci si trova a livello BASIC il comando di formattazione va impartito nel seguente modo:

(2) CMD"FORMAT"

Le virgolette finali possono essere omesse, e questo vale tutte le volte che esse si trovano alla fine di un comando o di una linea di programmazione; noi le metteremo sempre, per dare una correttezza anche formale a tutto quello che via via scriveremo, ma ricordati che nei casi appena detti esse possono essere tralasciate: si risparmia tempo nelle battute e, nel caso di programmi, si ottiene anche un risparmio di memoria.

Indipendentemente dal livello di partenza, dopo aver premuto il tasto RETURN vedrai partire i floppy: la procedura di formattazione ha avuto inizio.
Dopo poco lo schermo si cancella ed appare la scritta

(3)
```
NEDOS DISK FORMATTER PROGRAM 1.0
WHICH DRIVE IS TO BE USED ?

(programma 1.0 per formattare dischi)
(quale drive vuoi usare?)
```

A questo punto devi introdurre il numero del drive in cui hai inserito il disco vergine da formattare. Ovviamente, con due floppy, tale drive sara' il secondo, ossia quello contraddistinto dal numero 1; il primo drive, infatti, conterra' il disco NE-DOS (ricorda che il numero del primo drive e' zero!). Alla domanda precedente devi quindi rispondere scrivendo 1 e premendo RETURN (d'ora in poi ometteremo la continua ripetizione di premere il tasto RETURN, cosa abbastanza ov-

via).
Compare allora la scritta:

(4) DISKETTE NAME ?

(nome del dischetto?)

Devi quindi assegnare un nome al disco; se premi RETURN senza aver scritto nulla il sistema ti ripropone la domanda. Bisogna quindi dare almeno un carattere,

che puo' essere anche uno spazio, un numero od un simbolo qualsiasi. Se provi a scrivere piu' di otto caratteri, il computer li rifiuta, in quanto il nome del disco non deve superare quella lunghezza.
Dopo aver introdotto il nome, appare la dicitura

(5) CREATION DATE ?
(MM/DD/YY)

(data di formattazione?)
(mese/giorno/anno)

La data viene richiesta nella forma anglosassone, ossia prima il mese, poi il giorno e l'anno; ognuno di questi dati deve essere di due cifre e va separato da quello successivo col simbolo della barra trasversale. Se non si rispetta questa ultima regola, il computer richiede la data. Al posto delle cifre richieste possono essere battuti anche altri caratteri, come virgole o parentesi; e' pero' preferibile abituarsi ad introdurre la data in modo corretto, coi numeri appropriati: vedremo oltre che anche nel procedimento della copiatura dei dischi si deve specificare la data, ed in quel caso il sistema controlla l'attendibilita' dei dati digitati; meglio quindi prendere subito l'abitudine giusta.
Ecco qualche esempio di date corrette:

(6) 02/08/82 (8 febbraio 1982)
10/23/81 (23 ottobre 1981)
11/11/11

L'ultima, pur non avendo molto senso, e' formalmente corretta e veloce a digitarsi. Nel caso che il numero delle cifre non sia uguale a due per ogni elemento che costituisce la data, questa viene richiesta.

Dopo aver introdotto la data, si legge:

(7) MASTER PASSWORD ?

(parola chiave principale?)

Bisogna ora dare una parola, che dovrebbe servire per bloccare eventualmente l'accesso ai dati contenuti nel disco; in realta' tale possibilita' non e' operativa, in quanto si e' voluto fornire un sistema che non presentasse alcun tipo di protezione, per avere un computer trasparente e senza trucchi.
Puoi quindi dare un nome a caso, o battere semplicemente uno spazio; se premi solo RETURN il calcolatore ti rifa' la domanda.

Compare ora questo messaggio:

(8) DO YOU WANT TO LOCK OUT
ANY TRACKS ?

(vuoi inibire)
(qualche traccia?)

Se la risposta e' Y (da YES = si'), il computer pone subito dopo la seguente do-

manda:

(9) WHICH TRACKS (1-39) ?

(quale traccia da 1 a 39?)

ossia vuole sapere quale o quali tracce vanno eventualmente escluse; dopo aver fornito i dati richiesti, il procedimento prosegue con la domanda

(10) FORMAT THE LOCKED OUT TRACKS ?

(vuoi formattare le tracce inibite?)

La risposta deve essere Y o N. Non devi preoccuparti di questa parte del procedimento di formattazione: normalmente infatti si vuole sfruttare l'intero disco, quindi alla domanda (8) si risponde con la pressione del tasto N o di un altro qualsiasi (la barra dello spazio, ad esempio). Subito dopo appare la dicitura

(11) DIRECTORY WILL BE PLACED
ON TRACK 17

(l'indice sara' messo)
(nella pista 17)

Ora, finalmente, la formattazione ha inizio; sul monitor si puo' leggere:

(12) FORMATTING TRACK NN

(sto formattando la traccia NN)

dove il numero NN di due cifre parte da 00 e cresce di una unita' per volta, fino ad arrivare a 39: il sistema ci segnala il numero della traccia che sta formattando. Raggiunto il numero 39, sul video appare questa scritta:

(13) VERIFYING TRACK NN , SECTOR MM

(sto verificando la traccia NN, settore MM)

Il computer sta verificando se la formattazione ha avuto buon esito per tutte le tracce e tutti i settori; i numeri NN ed MM variano rispettivamente da 00 a 39 e da 00 a 09.

Se tutto e' regolare si puo' ora leggere:

(14) INITIALIZING SYSTEM INFORMATION
FORMATTING COMPLETE

(informazione del sistema sulla inizializzazione)
(formattazione completata)

La formattazione e' quindi terminata, ed ora quel dischetto e' pronto a ricevere dati.

Vediamo ora qualche esercizio.

ESERCIZIO n. 1

Quante sono le tracce ed i settori in cui viene suddiviso il dischetto con la formattazione?

.....
.....
.....
.....

RISPOSTA

Le tracce sono 40 ed i settori sono 10.

ESERCIZIO n. 2

Qual'e' il modo piu' breve, partendo dal livello BASIC, per chiedere una formattazione?

.....
.....
.....
.....

RISPOSTA

Dal BASIC, il modo piu' conciso per impartire il comando di formattazione e' il seguente: CMD"FORMAT.

ESERCIZIO n. 3

Come si deve indicare la data 3 AGOSTO 1982 ?

. . . . . .

. . . . . .

. . . . . .

. . . . . .

RISPOSTA

La data richiesta va scritta 08/03/82; le prime due cifre sono per il mese, le

seconde per il giorno; i tre numeri devono essere formati da due cifre.

---

3 - SEGNALAZIONI D'ERRORE IN FORMATTAZIONE -

Puo' succedere, durante l'operazione di verifica della formattazione, che il sistema scopra che una traccia e' difettosa; in questo caso, supponendo ad esempio che cio' accada nel quinto settore della traccia 23, il computer manda questo messaggio:

```
(15)      VERIFYING TRACK 23 , SECTOR 05
          NOT FOUND !  TRACK LOCKED OUT !

          (sto verificando la traccia 23, settore 05)
          (non l'ho trovata! Traccia inibita!)
```

e poi prosegue fino alla fine. Il messaggio ci informa che la traccia indicata risulta difettosa, e che non e' disponibile per la registrazione; il disco e' utilizzabile ugualmente, ma con una capienza leggermente ridotta. Il solo caso in cui non possiamo utilizzarlo e' nel procedimento di copia, che vedremo tra poco: non sarebbe infatti possibile fare una copia esatta di un disco usandone uno che e' difettoso in qualche traccia.

Se ti succede di trovare un dischetto formattato in questo modo incompleto, puoi provare a rifare la formattazione: spesso tutto torna a posto, in quanto il difetto era dovuto ad un granello di polvere che si era interposto tra testina e disco in registrazione od in lettura. Capita talvolta che rifacendo la formattazione in continuazione, ad un certo punto essa venga corretta: l'impurita' e' stata rimossa. Anche provando a cambiare il drive su cui si fa la formattazione puo' portare ad un risultato corretto.

Riallacciandoci a questo ultimo discorso, c'e' da dire che la formattazione si puo' fare anche se si ha a disposizione un solo floppy. In tal caso, naturalmente, si impartisce il comando (1) o (2) sempre col disco sistema inserito nell'unico drive disponibile (che sara' pertanto il numero zero). Al comparire della domanda (3), bisogna prima togliere il disco DOS-BASIC dal drive e sostituirlo con quello vergine da inizializzare. Si procede poi come nel caso precedente, premendo il tasto dello zero e RETURN, e cosi' via con tutto il resto. Alla fine comparira' il seguente messaggio:

```
(16)        INITIALIZING SYSTEM INFORMATION
            FORMATTING COMPLETE
            PRESS "ENTER" WHEN **SYSTEM**
            DISKETTE MOUNTED ON DRIVE 0

            (informazione del sistema sulla inizializzazione)
            (formattazione completata)
            (premere RETURN quando il disco)
            (sistema e' montato sul drive 0)
```


Al termine della formattazione si deve, in questo caso, rimontare il disco contenente il DOS e il BASIC nel drive, e poi, dopo avere premuto RETURN, si torna al livello da cui si era partiti. Queste sono le uniche differenze dal caso dei due floppy.

Vediamo ora quali messaggi manda il calcolatore se qualcosa non fila a dovere. Cercando di formattare un disco che abbia il nastro adesivo di protezione contro le registrazioni, si legge:

```
(17)        WRITE PROTECTED DISKETTE !
            REPLY "N" TO END FORMAT.
            REPLY "Y" AFTER ERROR CONDITION
            CORRECTED TO EXECUTE FORMAT
            COMMAND AGAIN

            (il disco e' protetto contro la scrittura!)
            (rispondi N se vuoi uscire dalla formattazione)
            (rispondi Y dopo aver rimediato all'errore)
            (per eseguire il comando)
            (di formattazione nuovamente)
```

Basta quindi togliere il nastro di protezione dal disco ed inserirlo nuovamente nel drive; introducendo la lettera Y si prosegue nella formattazione, mentre con la lettera N si esce dalla procedura e si torna al livello (DOS o BASIC) di partenza.

Anche un disco che contenga dei dati puo' essere formattato: e' quello che si fa normalmente quando il suo contenuto non ci interessa piu', poiche' CON L'INIZIALIZZAZIONE SI CANCELLANO TUTTI I DATI. Attenzione quindi a non farla su un disco che contenga files importanti, di cui non abbiamo una copia. Il computer, comunque, ci avvisa del fatto che stiamo per formattare un disco che non e' vergine col seguente messaggio:

```
(18)        FORMAT REJECTED,
            DISKETTE CONTAINS DATA !
            DISKETTE TO BE FORMATTED ANYWAY
            (REPLY "Y" OR "N")?

            (formattazione respinta)
            (il disco contiene dati!)
            (vuoi formattare ugualmente il disco?)
            (rispondi Y oppure N)
```

Premendo il tasto N si pone termine alla formattazione, salvando cosi' il contenuto del disco; se invece si batte Y si procede, cancellando tutto. E' buona norma, comunque, tenere il nastro di protezione sui dischi importanti.

Ancora: se la velocita' del drive non risulta essere quella giusta, il sistema lo segnala scrivendo:

```
(19)      CAN'T FORMAT,
          MOTOR SPEED TOO FAST (SLOW) !
```



```
          REPLY "N" TO END FORMAT.
          REPLY "Y" AFTER ERROR CONDITION
          CORRECTED TO EXECUTE FORMAT
          COMMAND AGAIN

          (non posso formattare!)
          (velocita' del motore troppo alta (bassa) !)
          (rispondi N se vuoi uscire dalla formattazione)
          (rispondi Y dopo aver rimediato all'errore)
          (per eseguire il comando)
          (di formattazione nuovamente)
```

In tal caso, che a dire il vero si verifica con estrema rarita', occorre rifare la taratura della velocita' del floppy, con l'aiuto dello stroboscopio che si trova sul suo volano (ovviamente bisogna smontare il drive dal mobile).

E c'e' dell'altro! Se non e' stato chiuso lo sportello del drive oppure se non e' stato inserito il dischetto, il computer segnala una o entrambe le condizioni dette con la scritta:

```
(20)      DOOR NOT CLOSED ON DRIVE !
          NO DISKETTE IN DRIVE !
          REPLY "N" TO END FORMAT.
          REPLY "Y" AFTER ERROR CONDITION
          CORRECTED TO EXECUTE FORMAT
          COMMAND AGAIN

          (lo sportello del drive non e' chiuso!)
          (non c'e' dischetto nel drive!)
          (rispondi N se vuoi uscire dalla formattazione)
          (rispondi Y dopo aver rimediato all'errore)
          (per eseguire il comando)
          (di formattazione nuovamente)
```

Come vedi, non gli sfugge proprio nulla! Al solito, dopo aver rimediato alla condizione d'errore, basta premere Y per proseguire.

Ancora una cosa, a livello di curiosita'; se alla domanda (8) si risponde con Y, ed alla (9) si replica col numero 17 (traccia dove normalmente viene messa la directory (ossia l'indice) del dischetto, essa verra' destinata altrove e tale nuova sistemazione viene scritta sul monitor con la segnalazione (11).
Infine, se alla domanda (8) si risponde Y e alla (9) si scrive:

(21) 0-39

si ordina al computer di inibire tutte le tracce dalla 0 alla 39; e' evidente che in tal caso il computer non sa dove andare a mettere la directory e lo segnala con la scritta:

(22) CAN'T FORMAT
NO TRACKS AVAILABLE FOR
DIRECTORY !

REPLY "N" TO END FORMAT.
REPLY "Y" AFTER ERROR CONDITION
CORRECTED TO EXECUTE FORMAT
COMMAND AGAIN

(non posso formattare!)
(non ci sono tracce disponibili per)
(la directory!)
(rispondi N se vuoi uscire dalla formattazione)
(rispondi Y dopo aver rimediato all'errore)
(per eseguire il comando)
(di formattazione nuovamente)

Si deve quindi rispondere, in questo caso, con N e poi ripartire da capo.

Come vedi, la casistica e' abbastanza ampia, ma anche molto chiara da seguire in quanto il sistema manda sempre segnalazioni che sono in relazione al tipo di errore o di inconveniente verificatosi.

Ricorda poi una cosa importante: NON FARE MAI UN BREAK A FORMATTAZIONE AVVIATA; PERDERESTI IL CONTROLO DEL SISTEMA E L'UNICO MODO PER RIPARTIRE E' QUELLO DI FARE UN RESET, PERDENDO IL BASIC E L'EVENTUALE PROGRAMMA IN MEMORIA.

4 - LA FORMATTAZIONE IN BREVE -

Tutto quello che si e' detto fin qui sulla formattazione ha portato via molto spazio, ma in effetti il procedimento e' assai semplice e veloce; il lavoro del floppy dura circa 50 secondi, cui va sommato il tempo necessario per rispondere alle varie domande. Vediamo di fare un riassunto delle operazioni da compiere per formattare un disco vergine, supponendo di avere due floppy e di mettere sul secondo il disco da inizializzare:

1 - introdurre il comando (1) o (2), a seconda del livello da cui si parte
2 - rispondere 1 alla domanda (3)
3 - dare un nome in risposta alla domanda (4)
4 - introdurre la data, alla domanda (5)
5 - battere la barra dello spazio alla domanda (7)
6 - rispondere N alla domanda (8)

Fatto questo, cosa che richiede pochi secondi, la procedura parte e in meno di un minuto avrai il disco inizializzato.

ESERCIZIO n. 4

Quali sono le possibilita' errate nell'elenco seguente?

1 - FORMATTARE UN DISCO CHE NON SIA VERGINE

2 - FORMATTARE UN DISCO COL NASTRO DI PROTEZIONE

3 - FORMATTARE SOLO ALCUNE TRACCE

4 - FARE UN BREAK DURANTE LA FORMATTAZIONE

5 - INIBIRE LA TRACCIA ZERO

6 - SCRIVERE LA DATA **/**/**

.....
.....
.....
.....

RISPOSTA

Le proposte sbagliate sono: 2-4-5. Per quest'ultima, basta osservare la (9); la pista zero e' riservata al sistema.

La lezione e' terminata. Dopo il riepilogo, esercitati a lungo nella formattazione; oltretutto faresti bene ad avere sempre qualche disco formattato di scorta. Puo' infatti accadere di non poter registrare un programma perche' il comando FORMAT non gira in BASIC se la memoria residua e' troppo bassa. Meglio quindi essere previdenti.

RIEPILOGO DELLA LEZIONE 5

FORMATTAZIONE

La formattazione (o inizializzazione) serve per poter utilizzare dischi vergini. Cercando di registrare su un disco non formattato si ha l'errore 58.

FORMAT

Il comando FORMAT serve per avviare la procedura di formattazione.

```
Es.     FORMAT          (a livello DOS)
Es.     CMD"FORMAT"     (a livello BASIC)
```

L E Z I O N E n.6

INDICE

1 - COPIATURA DEI DISCHI -

Dopo aver affrontato, nella lezione precedente, la formattazione dei dischi vergini, vediamo ora un'altra operazione molto importante: la copiatura del contenuto di un disco. La procedura normale di lavoro coi dischi presuppone sempre di averne almeno due copie uguali; possono infatti capitare diversi inconvenienti durante il loro uso. Citiamone qualcuno alla rinfusa: il disco cade e si rovina, oppure viene unto toccandolo con le dita, o irrimediabilmente piegato appoggiandoci sopra qualcosa; mentre lo stiamo usando, viene a mancare la tensione elettrica, con perdita del vecchio archivio e col nuovo rimasto a meta'; per qualche motivo una traccia si rovina, impedendo cosi' anche una ulteriore copia del disco in questione, e cosi' via.

E' quindi estremamente consigliabile avere sempre un disco di scorta, che verra' tenuto costantemente aggiornato mano a mano che l'originale cambia. A tale proposito c'e' da osservare che la copia risulta essere perfettamente identica all'originale: ogni eventuale errore di copiatura viene scoperto dal computer men-

tre questa e' in corso, ragion per cui, almeno in teoria, e' da escludere ogni difetto in questo senso; la distinzione tra originale e copia e' quindi solo nominale, poiche' e' impossibile distinguere i due dischi l'uno dall'altro. In casi come questo si parla di 'copia fisica' di un disco, proprio per indicare che non esistono blocchi elettronici, parole chiave od artifici di alcun genere che possano impedire l'esecuzione di una copia o che possano renderla diversa dall'originale.

Ovviamente la prima copia da eseguire, visto che la cosa si puo' fare, e' quella del disco sistema, ossia di quello contenente il DOS ed il BASIC; e' saggio poi usare correntemente la copia, conservando pressoche' integro l'originale per altre eventuali operazioni di duplicazione (che in linguaggio informatico viene comunemente indicata col nome di BACKUP).

Ribadiamo ancora l'importanza di avere non solo una copia dei dischi importanti, ma di tenerli anche aggiornati. Tenendo ad esempio una contabilita' per una ditta, o controllando i clienti di uno studio dentistico, praticamente ogni giorno vengono fatti degli interventi sui files-archivio, quindi conservare una copia vecchia anche solo di qualche giorno non avrebbe alcun senso: alla fine di ogni giornata, od anche piu' spesso per grosse moli di lavoro, e' assolutamente consigliabile effettuare una copia del disco aggiornato.

Finora abbiamo parlato di copia di un intero disco; e' comunque possibile farsi un duplicato anche solo di un file, sia esso un programma od un archivio. Questa operazione e' possibile perfino avendo a disposizione un solo drive, come vedremo tra poco. Non e' detto che si debbano sempre fare dei duplicati di dischi interi: se l'archivio aggiornato non e' molto grande conviene riversare solo quello da un disco ad un altro, con un notevole risparmio di tempo. Infatti la copia di un disco completo impiega circa sei minuti, mentre quella di un solo file e' piu' breve, a meno che il file stesso non occupi tutto il disco.

Dopo aver chiarito l'importanza che riveste il procedimento di copia, passiamo all'esame del relativo comando.
Al contrario di FORMAT, il procedimento di copiatura e' molto semplice e non presenta difficolta' rilevanti, specialmente se si opera con due floppy.

2 - COPY - (COPIA)

Anche il comando COPY, come FORMAT, appartiene al DOS; quindi per darlo in via diretta occorre portarsi a quel livello. L'operazione di duplicazione e', come al solito, possibile anche dal livello BASIC, facendo uso di CMD.

La procedura di duplicazione di un intero disco e' diversa a seconda che si abbiano a disposizione due floppy od uno solo. Parleremo del primo caso, che e' il piu' frequente; tra l'altro coloro che seguono questo corso hanno come minimo una tale configurazione. Dopo aver descritto la procedura da adottare con due o piu' floppy, diremo anche come si fa con uno solo.

Diamo il formato completo del comando COPY, supponendo di essere a livello DOS:

(1) COPY :0 TO :1 06/27/83

Se invece si parte dal BASIC si deve scrivere:

(2) CMD"COPY :0 TO :1 06/27/83"

con la solita osservazione sulle virgolette finali.
A parte quindi l'uso di CMD, il resto e' perfettamente uguale; parleremo allora solo del caso (1).

Andiamo con ordine. GLI SPAZI INDICATI SONO ASSOLUTAMENTE INDISPENSABILI; se essi vengono aboliti o se ne vengono dati due invece di uno, il computer rifiuta il comando con questa segnalazione d'errore:

(3) BAD FILESPEC

(errore di procedura)

Quello fornito e' un esempio scelto a caso; con esso si chiede di copiare il disco montato nel drive zero, trasferendo il suo contenuto nel disco che si trova nel drive uno. Deve poi seguire l'indicazione della data, scritta nel sistema anglosassone (mese, giorno, anno).
Contrariamente a quanto avviene nel comando FORMAT visto nella lezione 5, qui il computer esercita dei controlli non solo sulle barre trasversali, ma anche sui caratteri della data.

Ad esempio, provando a scrivere date come le seguenti

(4) 6/27/83
27/06/83
../../..
DF/11/AC

si ha questa segnalazione d'errore:

(5) BAD DATE

(errore di data)

Infatti nella prima di esse il mese 6 e' dato con una sola cifra, invece che 06; nel secondo esempio si e' messo prima il giorno, evidentemente, ed il sistema si rifiuta di prendere in considerazione un mese con numero 27. Infine, se, come nel terzo e quarto esempio, si introducono caratteri diversi da cifre, si ricade ancora in errore.

Assegnando invece il comando COPY in modo corretto, poco dopo si ottiene sul monitor la seguente scritta:

(6) PRESS "ENTER" WHEN SOURCE
DISKETTE MOUNTED ON DRIVE 0

(premere RETURN quando il disco sorgente)
(e' montato nel drive 0)

Si mette quindi il disco sorgente, ossia quello da duplicare, nel drive zero e poi si preme il tasto RETURN. Sullo schermo del monitor si legge allora:

(7) PRESS "ENTER" WHEN DEST.
DISKETTE MOUNTED ON DRIVE 1

(premere RETURN quando il disco)
(destinatario e' montato nel drive 1)

Si mette allora nel drive 1 un disco formattato e si preme RETURN. Immediatamente inizia il processo di copiatura. Appare infatti la seguente dicitura:

(8) READING TRACK, SECTOR NN,MM

(sto leggendo la traccia NN, settore MM)

Il led del drive 0 si accende, per indicare che sta avvenendo uno scambio di dati tra disco e memoria del computer. NN parte da 00, ed MM assume velocemente i valori tra 00 e 09: viene cioe' letto l'intero contenuto della prima traccia del disco sorgente, e quei dati vengono caricati in memoria. Il led poi si spegne, e contemporaneamente si accende quello del drive 1. La scritta precedente viene modificata in questo modo:

(9) WRITING TRACK, SECTOR NN,MM

(sto scrivendo la traccia NN, settore MM)

con le stesse considerazione sui numeri NN ed MM. Quando MM arriva a 09, compare la scritta:

(10) VERIFYING TRACK, SECTOR NN,MM

(sto verificando la traccia NN, settore MM)

A questo punto, quindi, il computer controlla che i dati incisi sul disco destinatario siano coincidenti con quelli del disco sorgente; per tal motivo abbiamo

detto poco fa che e' estremamente improbabile che una copia non sia conforme all'originale. Questo procedimento (che comporta prima una lettura, poi una registrazione, poi una lettura di quanto si e' inciso ed il suo confronto con l'originale) porta via molto tempo ma fornisce delle ottime garanzie sulla bonta' del risultato finale.

Tutto quello che e' stato descritto finora, valido per la copiatura della traccia 00, si ripete pari pari per ognuna delle 40 tracce: il processo ha quindi termine quando e' stato verificato il contenuto della traccia 39.
Compare a questo punto la scritta:

(11) FULL DISKETTE COPY DONE
PRESS "ENTER" WHEN SYSTEM
DISKETTE MOUNTED ON DRIVE 0

(la copia di tutto il disco e' finita)
(premere RETURN quando il disco)

(sistema e' montato nel drive 0)

A duplicazione finita si deve togliere il disco sorgente che era stato messo nel drive 0, inserendo al suo posto il disco sistema; ovviamente tale operazione e' superflua se il procedimento di copiatura e' stato fatto proprio per il disco del DOS-BASIC, in quanto esso si trova gia' nel drive 0.

Dopo la pressione del tasto RETURN si ha il ritorno al livello da cui eravamo partiti ed il sistema e' pronto per eseguire nuove istruzioni. Come abbiamo gia' detto, tutto il procedimento dura all'incirca 6 minuti ed avviene in modo completamente automatico, senza che necessiti, una volta che e' stato avviato, dell'intervento dell'operatore.

ESERCIZIO n. 1

E' corretto il comando COPY dato in questo modo?

COPY :O TO :1 12/12/12

.....
.....
.....
.....

RISPOSTA

Il comando non e' corretto, perche' e' stata scritta la lettera O al posto del numero 0, subito dopo i primi due punti; i due tasti O e 0 sono molto vicini, e puo' succedere di invertirli. In tal caso si ha la segnalazione d'errore (3).

In precedenza avevamo detto che e' possibile fare dei duplicati anche avendo a disposizione un drive solamente. Vediamo anche tale procedimento.

3 - COPIATURA CON UN SOLO DRIVE -

Se si dispone di un solo drive, l'operazione di duplicazione dei dischetti puo' essere fatta ugualmente; vediamo brevemente le differnze dal caso precedente.

Il comando va impartito nel seguente modo:

(12) CMD"COPY :0 TO :0 07/01/82"

dove la data e' puramente indicativa, come sempre. Il comando vale se si parte dal livello BASIC; dal DOS si tralasciano CMD e virgolette.
Se il comando e' dato in formato corretto, appare la scritta (6); dopo aver messo il disco sorgente nel drive 0 ed aver premuto RETURN si legge la scritta (8), la quale ci dice che il computer sta leggendo il disco sorgente. Infatti il led del drive e' acceso, ed i numeri NN ed MM variano nel modo che abbiamo visto poco fa. Stavolta pero' non viene letta solo la prima traccia, ma una porzione di disco sorgente piu' ampia: la quantita' esatta non e' fissa, ma dipende dalla

memoria disponibile. Con 56 K viene letto un quarto di disco.

Quando la lettura della porzione di disco sorgente e' finita, si legge la scritta (7), dove pero' al posto del numero finale 1 c'e' lo zero, come e' logico che sia, dal momento che si utilizza solo quel drive. Si deve allora togliere il disco sorgente ed inserire al suo posto quello destinatario; dopo la pressione del tasto RETURN, il led si riaccende, e compaiono successivamente le scritte (9) e (10): il sistema sta travasando tutti i dati letti e mano a mano che procede li controlla.

Quando l'intera porzione letta dal disco sorgente e' stata registrata su quello destinatario, riappare la scritta (6): il computer ci chiede di rimettere il disco da duplicare nel drive. Cio' perche' deve leggere la porzione successiva di dati; infatti dopo la scritta (8) riappare ancora la (7), poi la (9) e la (10), ossia tutto si ripete come descritto sopra.

Nel caso di un solo drive il procedimento e' un po' piu' laborioso, come si vede. A copia ultimata compare la scritta (11).

ESERCIZIO n. 2

Puo' essere impartito il seguente comando?

CMD "COPY :1 TO :1"

.....
.....
.....
.....

RISPOSTA

Il comando di COPY indicato e' lecito: anche avendo piu' di un floppy e' possi-

bile effetuare duplicazioni utilizzando un solo drive.

---

4 - SEGNALAZIONI D'ERRORE IN COPIATURA -

Vediamo ora quali sono gli errori che possono avvenire durante la procedura di duplicazione di un dischetto.
Si e' gia' detto degli errori che riguardano gli spazi da rispettare e le norme per la data: a questi due casi corrispondono i messaggi (3) e (5) di pagina 4.

Oltre a questi, si possono avere anche altri segnali d'errore. Se si tenta di fare una duplicazione su un disco destinatario che porti il nastro di protezione contro le registrazioni, si ha il seguente messaggio:

(13) PRESS "ENTER" WHEN SYSTEM

DISKETTE MOUNTED ON DRIVE 0

(premere RETURN quando il disco)
(sistema e' montato sul drive 0)

Dopo aver messo il disco DOS-BASIC nel drive 0 ed aver premuto RETURN, si legge:

(14) ILLEGAL FILE NAME

(nome di file illecito)

Tale segnalazione, in tal caso, significa che la manovra e' illecita proprio per la presenza dell'adesivo di protezione. Per procedere lo si deve rimuovere, oppure cambiare disco. In ogni caso, il comando di duplicazione va impartito nuovamente.

Se ti succede, poi, di fare una copia mettendo come destinatario un disco che ha qualche traccia inibita (o perche' l'avevi inibita tu stesso, o a causa di un difetto del disco, come abbiamo visto nella lezione precedente a pagina 8), arrivato a quel punto il computer segnala errore prima con la scritta (13), poi, dopo aver rimesso il disco sistema nel drive 0 ed aver premuto RETURN, si ha la seguente dicitura:

(15) PARITY ERROR DURING READ

(errore di parita' durante la lettura)

Provando invece ad aprire lo sportello del drive mentre in esso e' in corso la registrazione dei dati , si ha, ad esempio:

(16) WRITING TRACK, SECTOR 03,0BAD
PARAMETER

(sto scrivendo la traccia 03, settore 0 parametro)
(sbagliato)

Tale manovra e' comunque al limite della sopportazione per il computer: spesso, comportandosi in quel modo, si perde il controllo del sistema e bisogna premere il tasto del RESET.

Se nell'impartire il comando di copia si da' un numero di drive impossibile, come in questo caso:

(17) CMD"COPY :1 TO :5 04/18/83"

si ottiene la segnalazione d'errore seguente:

(18) BAD FILE NAME IN 15359

Qualora si impartisca questo comando:

(19) COPY :0 TO :2 10/02/82

e si abbiano solo due floppy, il computer perde quasi sempre il controllo e si riparte solo con un RESET.

Per concludere le considerazioni sul procedimento di copiatura dei dischi, facciamo notare che se il disco destinatario non e' formattato il computer esegue regolarmente la parte iniziale del processo, ma quando arriva alla scrittura dei dati resta fermo per un po' e poi manda la segnalazione (13); dopo aver premuto RETURN compare la scritta:

(20) DIRECTORY READ ERROR

(errore nella lettura della directory)

C'e' da notare, infine, che se il disco destinatario contiene dei dati, non si ha alcuna segnalazione del fatto: la copiatura procede fino in fondo, e tutti i dati precedenti vengono naturalmente sostituiti dai nuovi. Occorre quindi accertarsi sempre che il disco destinatario non contenga dati importanti.

5 - COPIATURA DI UN FILE -

Spesso torna comodo trasferire da un disco ad un altro un file, sia esso un programma od un archivio di dati. Mentre nel primo caso si potrebbe anche caricare il programma nella memoria centrale del computer e poi effettuare il suo salvataggio su di un altro disco, nel caso di un file di dati la cosa non e' possibile, a meno che non si faccia un apposito programma per leggerlo e per trasferirlo successivamente sul nuovo dischetto.

Facendo uso del comando COPY e' altresi' possibile trasferire files da un disco ad un altro. Supponiamo di voler travasare il file di nome INDICE dal dischetto montato sul drive 1 a quello del drive 0; il comando da impartire e' questo:

(21) COPY INDICE:1 TO :0

Dopo averlo impartito, il computer si mette all'opera e preleva i dati dal se-

condo drive per registrarli sul primo. Questo procedimento puo' avvenire una sola volta o piu' volte, a seconda della lunghezza del file.

Nel fare il trasferimento si puo' anche cambiare il nome; supponendo di copiare il file precedente assegnandogli anche il nuovo nome ELENCO, si deve scrivere:

(22) COPY INDICE:1 TO ELENCO:0

Al solito, il rispetto degli spazi e' condizione indispensabile affinche' il comando venga accettato.

Qualora sul disco destinatario esista gia' un file di nome uguale a quello specificato, si ha la seguente segnalazione d'errore:

(23) SOURCE & DEST SAME FILE
UNPRINTABLE ERROR IN 15359

(stesso file sorgente e destinatario)
(errore non stampabile in 15359)

Questa segnalazione si legge anche se si cerca di copiare un file sul medesimo disco in cui si trova; anche in questo caso, come nel precedente esiste un rimedio. Esso consiste nel cambiare nome al file. Ad esempio:

(24) COPY INDICE:1 TO ELENCO:1

e' ammesso ed eseguito; il risultato finale sara' quello di avere sul dischetto montato sul drive 1 due files di identico contenuto, ma con due nomi differenti.

Il comando di COPY di un file puo' essere anche scritto nel seguente formato:

(25) COPY INDICE:1 ELENCO:0

ossia tralasciando la parola TO: tutto procede ugualmente bene. Tale semplificazione non va invece fatta nel caso della duplicazione di un intero disco: si otterrebbe la segnalazione d'errore (3).

Ancora una curiosita': operando con un solo drive e' possibile perfino fare una copia di un disco sullo stesso disco; provare per credere! La cosa non ha molto senso, ma e' abbastanza indicativa del modo di procedere del sistema.

La lezione e' finita. Dopo il riepilogo, devi esercitarti a lungo nell'applicazione del comando COPY, eseguendo tutte le varianti che abbiamo appena visto; e' importantissimo per acquisire la necessaria dimestichezza con questa procedura cosi' importante.

La prima cosa che ti consigliamo di fare, come abbiamo detto, e' quella di eseguire delle copie del disco sistema, per metterti al riparo da possibili danneggiamenti. A tale riguardo, ricorda di togliere sempre i dischetti dai drive prima di spegnere il computer, altrimenti gli stessi potrebbero diventare inservibili. Osserva la stessa precauzione pure per l'accensione.

RIEPILOGO DELLA LEZIONE 6

COPY

COPY e' un comando DOS e puo' essere usato in due modi diversi: per eseguire duplicati di interi dischi, o per copiare solo dei files. Nel primo caso il formato per impartire il comando e' il seguente:

```
Es.     COPY :0 TO :1 08/29/82          (a livello DOS)
Es.   . CMD"COPY :0 TO :1 08/29/82"     (a livello BASIC)
```

Per fare invece la copia di un file si hanno i seguenti formati:

```
Es.     COPY ELISA:0 TO :1
Es.     COPY ELISA:0 :1
Es.     CMD"COPY ELISA:0 TO BARBARA:1
```

Es. CMD"COPY ELISA:O BARBARA:O

L E Z I O N E n.7

INDICE

1 - RENAME - (ASSEGNA UN NUOVO NOME)

Continuiamo nell'esame dei comandi utili alla gestione dei dischi e dei files in particolare; iniziamo da RENAME. Esso serve per assegnare un nuovo nome ad un file gia' presente su disco. Dal momento che si tratta di un comando DOS, valgono le solite considerazioni su CMD: partendo dal BASIC, esso va specificato, mentre dal DOS deve essere tralasciato.

Supponendo di essere al livello BASIC, il formato di RENAME e' il seguente:

```
(1)       CMD"RENAME MEDEA:1 TO MARIA"
```

In questo esempio si ordina di prendere il file di nome MEDEA, che si trova nel disco inserito nel drive 1, e di assegnargli il nuovo nome MARIA.

Anche il seguente comando ottiene lo stesso effetto:

```
(2)       CMD"RENAME MEDEA MARIA"
```

Le differenze rispetto al caso precedente sono due: la prima consiste nella mancanza dei due punti e del numero successivo; non avendo specificato su quale drive va esercitato il comando, il computer lo esegue ugualmente, esplorando pero' il contenuto dei dischi inseriti nei vari drives; la ricerca parte dal drive numero zero, e se in quel disco non esiste il file MEDEA, si ha il passaggio sul drive uno, e cosi' via. Il comando viene effettivamente eseguito quando il file e' stato trovato.
La seconda differenza consiste nell'omissione della parola TO, che e' facoltativa e puo' sempre essere tralasciata.

In ogni caso, sia che si usi un formato o l'altro, se il computer non trova il file indicato si ottiene la seguente segnalazione:

```
(3)       FILE NOT FOUND IN 15359

          (file non trovato)
```

La stessa scrittura appare anche se non viene messo lo spazio dopo RENAME.
Provando a mettere i due punti ed il numero del drive anche dopo il nome nuovo, si puo' leggere sul monitor:

```
(4)       DRIVE SPECIFICATION ILLEGAL
          UNPRINTABLE ERROR IN 15359

          (specifica di drive illecita)
          (errore non stampabile)
```

Con questo, abbiamo gia' detto tutto sul comando RENAME, per mezzo del quale e' immediato cambiare il nome di un file senza essere costretti a registrarlo nuovamente col nome diverso e a cancellare poi quello col nome vecchio.

La cancellazione di un file e' proprio l'argomento che affronteremo nel paragrafo che segue.

2 - KILL - (UCCIDI)

Il comando KILL serve per cancellare un file dal disco; dopo la sua esecuzione, il posto che prima essso occupava e' nuovamente disponibile per la registrazione di dati.

KILL e' l'unico comando che appartiene sia al DOS che al BASIC; puo' quindi essere impartito nei seguenti modi:

(5) KILL LUCA

(6) CMD"KILL LUCA"

(7) KILL"LUCA"

Il primo e' il comando KILL applicato a livello DOS; il secondo e' sempre il comando KILL del DOS, usato pero' partendo dal BASIC; il terzo e' invece il comando KILL del BASIC, che ottiene lo stesso effetto dei due precedenti, ma che non ha niente a che fare con essi.

Notare la differenza tra il primo e il terzo modo: a livello BASIC le virgolette sono indispensabili, altrimenti si ottiene la seguente segnalazione d'errore:

(8) TYPE MISMATCH IN 15359

(cattivo assortimento di caratteri)

Vediamo quali sono le altre segnalazioni d'errore.
Supponiamo innanzitutto di operare a livello DOS; se mettiamo piu' di uno spazio dopo KILL otteniamo:

(9) FILE SPEC REQUIRED

(e' richiesta la specifica del file)

Questa stessa segnalazione viene data scrivendo solo KILL, senza il nome del file da cancellare.

Se invece, ad esempio, scrivessimo:

KILL PROVA:6

commetteremmo un errore, poiche' ordineremmo al computer di cancellare il file di nome PROVA sul disco inserito nel drive numero sei, cosa questa impossibile dal momento che al massimo tale numero puo' valere 3; sul monitor infatti comparirebbe la dicitura:

(10) ILLEGAL DRIVE NUMBER

(numero di drive illecito)

Proseguiamo nell'esame dei possibili errori; eliminando lo spazio dopo KILL si

ha la seguente scritta:

(11) PROGRAM NOT FOUND

(programma non trovato)

E' successo che il computer interpreta il tutto come il nome di un file da cercare e, non trovandolo, manda quel messaggio.

Se il file richiesto non viene trovato su nessun drive, si puo' leggere:

(12) FILE NOT IN DIRECTORY

(file non compreso nell'indice)

Dopo aver visto le segnalazioni d'errore nel caso del DOS, vediamo quelle che si incontrano a livello BASIC. Bisogna distinguere due casi, a seconda che si sia fatto uso del formato (6) o del (7).
Mettendoci nel caso (6), cominciamo dalla scritta (9), che diventa:

(13) FILE SPEC REQUIRED
UNPRINTABLE ERROR IN 15359

(e' richiesta la specifica del file)
(errore non stampabile)

L'errore (10) diventa:

(14) BAD FILE NAME IN 15359

(cattivo nome di file)

La segnalazione (11) diventa ora:

(15) INTERNAL ERROR IN 15359

(errore interno)

La scritta (12) cambia in:

(16) FILE NOT FOUND IN 15359

(file non trovato)

Se, invece, siamo nel caso del formato (7), l'unico caso che non sia gia' stato esaminato e' il seguente:

KILL" PROVA"

ossia quello di lasciare uno o piu' spazi dopo le virgolette; sul monitor compare allora la scritta (14).
Naturalmente, oltre agli errori che abbiamo esposto, puoi anche scrivere in modo

diverso la parola KILL, a causa di un errore di battuta; in un caso del genere vedrai la scritta (11) se stai operando a livello DOS, mentre se ti trovi a livello BASIC hai:

```
(17)        SYNTAX ERROR IN 15359
            UNDEFINED LINE # IN 15359

            (errore di sintassi)
            (numero di linea non definito)
```

Crediamo cosi' di averti dato tutti i possibili errori; l'esposizione e' stata probabilmente un po' pedante, ma la completezza della trattazione lo richiedeva.

ESERCIZIO n. 1

Se hai tre drives ed in ognuno di essi e' inserito un disco contenente il file di nome TABELLA4, qual'e' il comando piu' breve per cancellare quel file dal disco che si trova nel terzo drive? (Partenza dal BASIC)

.....
.....
.....
.....

RISPOSTA

Partendo dal BASIC ed usando il comando KILL del BASIC (con quello del DOS il comando sarebbe piu' lungo), si deve scrivere:

```
KILL"TABELLA4:2
```

(In effetti tutta la scrittura precedente puo' essere in minuscolo; ci pensa il computer a trasformarla in maiuscolo).

ESERCIZIO n. 2

Supponi di dover spostare sul drive 0 un programma di nome CARLO che si trova sul drive 1, e di volergli anche cambiare nome, facendolo diventare CARLO5. Scegli, tra le soluzioni seguenti, quella che ti sembra la migliore, supponendo di partire dal livello BASIC:

```
1 - CMD"RENAME CARLO CARLO5"

2 - CMD"COPY CARLO5:1 TO :0"

3 - COPY CARLO:1 TO CARLO5 :0
```

4 - CMD"COPY CARLO:0 TO CARLO5:1"

5 - CMD"COPY CARLO:1 CARLO5:0

6 - CMD"COPY CARLO :1 TO CARLO5 :0"

.....
.....
.....
.....

RISPOSTA

La risposta giusta e' la 5: con essa si travasa il programma CARLO dal drive 1 al drive 0 cambiandogli contemporaneamente il nome in CARLO5. La soluzione 3 e' giusta solo se si parte dal livello DOS. La 4 e' sbagliata: i numeri dei due drives sono stati invertiti. La risposta 6 e' gravemente errata, a causa degli spazi che sono stati messi dopo i due punti.
La 1 e la 2 servono rispettivamente per cambiare di nome e per cambiare di disco al programma in questione; da notare che applicandole una dopo l'altra si ottiene sempre il risultato cercato, anche se per una via piu' lunga.
Il travaso del file avviene in una o piu' passate, a seconda della sua lunghezza.

---

3 - FREE - (LIBERO)

Proseguiamo nella rassegna dei comandi del DOS, prendendo in esame FREE.
Esso e' molto utile poiche' permette di conoscere quanto spazio ancora libero ci sia in un dischetto.

Ricordiamo velocemente la suddivisione operata dalla formattazione: 40 tracce, ognuna delle quali divisa in 10 settori.
Ebbene, quando il sistema deve registrare un file, esso comincia col riservargli 5 settori (tale quantita' e' chiamata GRANULO); se il file e' breve, esso sara' contenuto in tale spazio. Se si ricorda che un settore contiene 256 bytes (ossia caratteri), si puo' dire che l'ingombro minimo di un qualsiasi file, piccolo quanto si vuole, e' di 256*5=1280 bytes.
Nell'ipotesi, invece, che il file sia piu' lungo di 1280 bytes, ad esso viene riservato un altro granulo, e cosi' via fino a che tutto il file trova posto sul disco. Il granulo rappresenta quindi l'unita' di misura dello spazio occupato da un file.

I granuli a disposizione dell'utente in un disco appena formattato sono 77 e non 80 come potrebbe sembrare: 40 tracce, ognuna delle quali contenente due granuli. Infatti una traccia intera e' riservata alla directory, e un'altra mezza serve al sistema per leggere e scrivere correttamente sul dischetto. Abbiamo allora che in totale il sistema porta via 3 granuli; ecco quindi che gli 80 teorici si riducono a 77; l'effettiva capacita' di un disco e' percio' di 256*5*77=98560 bytes.

Dobbiamo ancora parlare di una cosa, ossia del numero massimo di files che possono trovar posto in un disco; e' evidente, infatti, che la traccia riservata all'indice non ha una capienza illimitata, e puo' contenere solo un certo numero di nomi. Tale limite e' 48.

Ora abbiamo tutti gli elementi che ci servono per poter parlare di FREE. Esso e' un comando DOS, quindi puo' essere impartito nei soliti due modi, a seconda del livello da cui si parte:

```
(18)      FREE                (dal DOS)

(19)      CMD"FREE"           (dal BASIC)
```

Come si puo' vedere, l'uso di FREE e' estremamente semplice ed immediato. Dopo aver ricevuto il comando, il sistema parte e presenta sul monitor una scritta di questo tipo:

```
(20)      DRIVE 0 --  NE-DOS     01/01/82
               44 FILES,    28 GRANS
```

Il numero di queste diciture dipende dal numero dei drives collegati e contenenti un disco. Ad esempio, con due drives avremo due di queste scritte, a patto che anche il secondo drive contenga un dischetto; infatti i drives vuoti o con lo sportello aperto vengono ignorati.

Analizziamo il contenuto della dicitura; si comincia col numero del drive, che potra' variare da 0 a 3. Seguono poi il nome assegnato al dischetto (NE-DOS nel nostro esempio) e la data di creazione: questi sono i dati assegnati in sede di formattazione o di copiatura del disco. Nella seconda riga troviamo il numero dei files ancora disponibili (ossia la differenza tra i 48 di partenza ed il numero di quelli gia' registrati); segue, infine, il numero dei granuli ancora liberi.
Evidentemente un disco pieno presenta 0 GRANS, ma in genere ha ancora un certo numero di files, sui 48 iniziali, non utilizzati.

Nell'uso di FREE puo' capitare un solo inconveniente, che e' quello di inserire in un drive un disco non formattato; in questo caso il computer tenta ripetutamente di leggere i dati che gli servono. Non trovandoli, ritorna al drive 0 per presentare la seguente scritta:

```
(21)      DISK I/O ERROR. USE E-CMD FOR SPECIFIC IN 15359

               (errore di input/output col disco)
               (usare CMD"E" per conoscere il tipo d'errore)
```

Seguendo l'istruzione ricevuta, cioe' digitando:

```
(22)      CMD"E"
```

vedremo il seguente messaggio d'errore:

```
(23)      GAT READ ERROR
```

(errore di lettura)

Provando a chiedere il codice dell'errore con la formuletta:

(24) ? ERR/2+1

si ottiene il numero 58 (vedi l'appendice N.2 a pagina 8 dell'introduzione).

Il concetto di granulo ritornera' quando parleremo diffusamente del comando DIR; cerca pertanto di assimilarlo bene.

ESERCIZIO n. 3

Se un file e' lungo 6422 bytes, quanti granuli occupa? (Usa una calcolatrice tascabile per fare piu' velocemente il conto!)
.....
.....
.....
.....

RISPOSTA

Per trovare la risposta esatta conviene prima cercare il numero dei settori occupati da 6422 caratteri: basta dividere 6422 per 256, ottenendo come risultato il numero 25.0859. Cio' significa che il file occupa 26 settori; dato che un granulo contiene 5 settori, sono necessari 6 granuli (ossia 30 settori) per contenere il file assegnato.
Si poteva arrivare direttamente al risultato con l'operazione:

6422/(256*5)=5.01719

Dal momento che il risultato e' maggiore di 5, il valore giusto e' l'intero successivo, ossia 6.

---

4 - LIB - (LIBRARY = BIBLIOTECA)

Il comando LIB, pure esso appartenente al DOS, e' estremamente semplice da usare e serve per avere l'elenco dei comandi DOS.

Il formato e' il seguente:

(25) LIB (dal DOS)

(26) CMD"LIB" (dal BASIC)

Il computer risponde a questo comando con la seguente scritta:

```
(27)      APPEND ATTRIB AUTO
          COPY   DATE   DIR    DUMP
          FREE   KILL   LIB    LIST
          LOAD   PRINT  PROT   RENAME
          TIME   VERIFY
```

Questo elenco corrisponde al contenuto della tabella (13) della lezione 4, pagina 9, che pero' presenta tre voci in piu': BASIC, DEBUG e FORMAT. Questi infatti non sono veri e propri comandi del DOS, ma i nomi di tre programmi che servono, rispettivamente, per caricare il BASIC, per passare in DEBUG (che vedremo in futuro) e per formattare i dischi. Tali programmi sono sul disco sistema e vengono agganciati automaticamente dal computer quando gli impartiamo uno di quei comandi.
Questo spiega la mancanza di quelle voci nell'elenco fornito da LIB.

La lezione e' finita. Come ti consigliamo sempre, dopo la stampa del riepilogo e' opportuno che tu ti eserciti a fondo nei comandi RENAME, LIB, KILL e FREE. Specialmente gli ultimi due sono di uso molto frequente, e non devi avere dubbi od esitazioni nella loro applicazione.

RIEPILOGO DELLA LEZIONE 7

RENAME

RENAME e' un comando del DOS e si usa per cambiare il nome di un file che sia gia' registrato su disco. Il formato e' il seguente:

```
Es.     RENAME RUBRICA ELENCO          (dal DOS)
Es.     CMD"RENAME RUBRICA ELENCO"     (dal BASIC)
```

KILL

Anche KILL e' un comando del DOS, e serve per cancellare un file dal disco.

```
Es.     KILL DREAM4                    (dal DOS)
Es.     CMD"KILL DREAM4"               (dal BASIC)
```

FREE

Il comando FREE del DOS serve per conoscere il numero dei granuli ancora liberi nei dischi inseriti nei vari drives.

```
Es.     FREE                           (dal DOS)
Es.     CMD"FREE"                      (dal BASIC)
```

LIB

LIB fornisce l'elenco dei comandi DOS riconosciuti dal sistema.

```
Es.     LIB                            (dal DOS)
Es.     CMD"LIB"                       (dal BASIC)
```

L E Z I O N E n. 10

INDICE

1 - SCRITTURA DI UN PROGRAMMA -

Finora abbiamo esaminato il funzionamento di alcuni comandi diretti e di qualche istruzione; dovresti inoltre avere abbastanza chiaro il concetto di variabile numerica.
E' giunto quindi il momento che tu ti cimenti nella scrittura di un piccolo programma; per cominciare staremo sul molto facile, in modo da non creare difficolta' insormontabili. Il programma che dovrai elaborare tra poco fornisce l'area di un triangolo rettangolo, dopo che l'operatore avra' introdotto il valore dei due cateti.

Fin dall'inizio dovrai abituarti ad una stesura ordinata e logica delle varie linee di programmazione; e' per questo motivo che l'esercitazione seguente e' stata concepita in modo abbastanza rigido. Infatti dovrai rispettare le seguenti regole:

- la numerazione delle linee deve iniziare da 10 e proseguire con passo 10;
- tutto quello che scrivi deve essere digitato in caratteri maiuscoli; se farai battute in minuscolo esse verranno rifiutate, ed il computer ti indichera' il primo carattere minuscolo che compare nella riga;
- devi lasciare uno spazio tra il numero di linea e la prima istruzione, pure tra ogni istruzione e cio' che la segue.

In pratica la forma da rispettare e' quella che hai vista applicata finora nei numerosi esempi gia' visti.

In effetti il computer non richiederebbe il rispetto cosi' rigoroso di tutte queste regole: gli spazi, come abbiamo detto in piu' di un'occasione, non sono determinanti per il corretto funzionamento di un programma. E' pero' consigliabile abituarsi a metterli regolarmente, ed in modo logico; solo cosi' il listato risultera' facilmente leggibile. Tieni presente, poi, che in molti computer questa procedura e' d'obbligo; in caso contrario il programma non girerebbe.

Anche l'obbligo di scrivere tutto in maiuscolo puo' sembrarti superfluo; esso e' stato messo perche' devi abituarti a distinguere le battute maiuscole da quelle minuscole, che sono indistinguibili sul monitor. Il computer, come abbiamo gia' fatto notare altre volte, e' pero' in grado di distinguere le due cose, come appare anche da un listato su stampante.
Tale discorso cade per i computer corredati della scheda grafica: in quel caso il sistema scrive sul video anche in minuscolo, quindi il problema non esiste piu'.

In ogni caso, per i prossimi esercizi, l'obbligo di digitare tutto in maiuscolo verra' tolto, tranne che nei casi specificati volta per volta.

La possibilita' di eliminare gli spazi puo' tornare utile per risparmiare celle di memoria in programmi molto estesi; fa' inoltre risparmiare tempo durante la scrittura alla tastiera, ma questi due fattori non sono certo significativi in questi primi esempi. Il consiglio che ti diamo e' quindi quello di scrivere tutti i tuoi programmi rispettando queste regole; l'ordine formale che ne deriva ti aiutera' nell'acquisizione del corrispondente rigore mentale, indispensabile a chi desidera dedicarsi seriamente alla programmazione in BASIC.

Il programma che devi scrivere contiene le seguenti istruzioni:

CLS PRINT INPUT GOTO

oltre all'uso di virgolette, punto e virgola, virgola, due punti e vari operatori aritmetici. Tieni quindi a portata di mano il testo delle lezioni fatte finora, in modo da poterle consultare facilmente; anzi, se non hai mai fatto una precedente pratica di stesura di programmi, e' meglio che tu ti rilegga la lezione 2, che tratta appunto di tutte le istruzioni che troverai tra poco.
In particolare, il programma (8) a pagina 9 ti puo' essere d'aiuto.

Ti accorgerai ben presto di non poter avanzare nella stesura del programma se non rispetti le regole menzionate; per ogni errore commesso il computer ti mandera' un messaggio appropriato. Seguendo passo passo le sue indicazioni arriverai a scrivere le sette righe del programma richiesto. Se proprio ti trovi in difficolta', nel senso che non riesci a scrivere il contenuto di una riga nel modo giusto, potrai conoscerne l'esatta formulazione e proseguire nella stesura di quelle seguenti introducendo, invece del contenuto della linea, un punto interrogativo. Resta sottinteso che per introdurre una linea devi premere il tasto RETURN, esattamente come se ti trovassi veramente in sede di programmazione.
Per ogni nuova linea di programma, troverai sul monitor alcune indicazioni utili alla sua corretta stesura; leggendo attentamente quei suggerimenti non dovresti incontrare difficolta' eccessive.

2 - INPUT (SECONDA PARTE) -

Nella terza linea del programma dovrai applicare l'istruzione INPUT in un modo che non abbiamo ancora visto. Per spiegarlo facciamo riferimento al programma (8) della lezione 2. Esso potrebbe essere scritto anche nel modo seguente:

(1)
```
10 'SOMMA DI DUE NUMERI
20 CLS
30 INPUT "PRIMO E SECONDO NUMERO" ; A,B
40 PRINT : PRINT A,B : PRINT : PRINT
50 PRINT "LA LORO SOMMA VALE : " ; A+B
60 PRINT : PRINT : PRINT : PRINT : GOTO 30
```

Se confronti questo programma con quello menzionato, ti accorgerai che la differenza consiste in questo: ora c'e' una sola istruzione INPUT, utilizzata per richiedere due valori numerici; la cosa funziona a patto di scrivere, dopo il punto e virgola che segue INPUT, gli identificatori delle due variabili, separati da una virgola. In pratica la linea 30 del programma appena proposto sostituisce le righe 30-40-50 di quello precedente.
Alle considerazioni fatte per INPUT alle pagine 7 e 8 della lezione 2, possiamo aggiungere che, qualora al punto interrogativo che compare sul monitor quando l'istruzione INPUT diventa operante non si risponda con un numero, appare la dicitura:

(2)
```
?REDO
```

Cio' significa che il computer non accetta quello che e' stato digitato al posto

del valore numerico richiesto.

Nel caso poi che si usi l'istruzione INPUT nel modo appena visto, se dopo aver introdotto il primo dato si preme RETURN, sul monitor appare:

(3) ??

Il doppio punto interrogativo sta a significare che bisogna introdurre altri dati. Facciamo notare che se, arrivati a questo punto, si introduce un valore non numerico, il computer manda ancora la segnalazione (2), cui fa seguito la richiesta del primo dato, come attesta anche il fatto che compare un solo punto interrogativo.

Per finire, precisiamo che con una sola INPUT possono essere richiesti diversi valori, come si vede dall'esempio seguente:

(4)
```
.....
580 INPUT G,C3,AA,KJ,F
.....
```

In questo caso sul monitor compare solo il punto interrogativo, dal momento che manca, dopo l'istruzione INPUT, la frase tra virgolette ed il relativo punto e virgola.

3 - ESERCIZIO -

Nell'esecuzione dell'esercizio che segue tieni quindi conto di quanto detto per INPUT, sia alla lezione 2 che poco fa.
Un'ultima cosa, molto importante: QUANDO AVRAI TERMINATO DI SCRIVERE TUTTO IL PROGRAMMA, IL COMPUTER VA SUL DISCO A REGISTRARLO. A lezione finita non dovrai quindi riscriverlo per provare ad usarlo, ma lo troverai gia' presente col nome AREA1; digitando quindi:

(5) RUN "AREA1"

potrai caricarlo in memoria ed eseguirlo. Scrivendo invece:

(6) LOAD "AREA1"

ti sara' facile listarlo su monitor o stampante, facendo uso, rispettivamente, dei seguenti comandi:

(7) LIST "AREA1"

(8) LLIST "AREA1"

Tieni pero' presente che IL COMPUTER NON PUO' EFFETTUARE REGISTRAZIONI SU DISCO QUALORA QUESTO CONTENGA IL NASTRO DI PROTEZIONE CONTRO LA SCRITTURA: se ti trovi in questa condizione, il programma e' fatto in modo che ti avvisa dell'inconveniente: basta allora estrarre il disco dal drive, togliere il nastro di protezione e poi proseguire con la pressione di un tasto.

Dopo aver terminato l'esercitazione, troverai la spiegazione dettagliata del programma.

A questo punto crediamo di averti detto tutto; passiamo quindi all'esercizio.

4 - SPIEGAZIONE DEL PROGRAMMA -

Ecco il listato del programma che hai appena scritto:

```
(9)       10 CLS
          20 INPUT "CATETI" ; C1,C2
          30 PRINT : ?                    (PRINT : PRINT)
          40 A=C1*C2/2
          50 PRINT "AREA=" ; A
          60 ? : ? : ?                    (PRINT : PRINT : PRINT)
          70 GOTO 20
```

In effetti quello riportato e' cio' che hai digitato: il listato, alle righe 30 e 60, mostra le parole chiave PRINT per esteso (come appare nelle parentesi), e non il punto interrogativo.
Vediamo la spiegazione delle linee del programma, una per una.

LINEA 10 - L'istruzione CLS cancella lo schermo del monitor (vedere la lezione 2, pagina 8, paragrafo 7);

LINEA 20 - Essa serve per richiedere all'operatore i valori numerici dei due cateti C1 e C2; L'istruzione usata e' INPUT (lezione 2, pagina 7, paragrafo 6). La frase tra virgolette ha lo scopo di chiarire quali sono i dati da digitare; in coda alla frase viene stampato automaticamente un punto interrogativo. I dati richiesti sono due, dal momento che il punto e virgola e' seguito da due identificatori di variabili numeriche, separati da una virgola (presente lezione, pagina 4, paragrafo 2). I dati introdotti dall'operatore vengono depositati in quelle variabili.
Da notare che C1 e C2 sono delle variabili numeriche in singola precisione, dal momento che non e' diversamente specificato: infatti non sono seguite ne' dal simbolo # (che le trasformerebbe in variabili in doppia precisione), ne' dal carattere % (variabili numeriche intere). L'uso del simbolo $ le avrebbe trasformate in stringhe. In tal caso il computer accetterebbe non solo numeri, ma anche altri caratteri.
Evidentemente, nel nostro caso, dobbiamo trattare dei numeri. Qualora sei cifre significative non siano giudicate sufficienti, occorre passare alle variabili in doppia precisione; esse ne fornirebbero 16.
A questo riguardo, rileggi quanto detto nella lezione 1, paragrafo 4, pagina 5 e seguenti.

LINEA 30 - Come abbiamo visto nella lezione 2 (pagina 4, paragrafo 2), la parola chiave PRINT, quando non e' seguita da variabili o stringhe, serve per lasciare righe vuote di video. Tale istruzione puo' essere abbreviata con un punto interrogativo; il listato del programma presentera' poi l'istruzione per esteso.

LINEA 40 - Qui compare la definizione della variabile numerica in singola precisione A, in cui viene depositato il valore dell'area del triangolo rettangolo che ha per cateti i valori di C1 e C2 introdotti dall'operatore e alla linea 20. Il simbolo * indica al computer che le due variabili vanno moltiplicate tra di loro. Il risultato di tale operazione viene diviso per due, come indica il simbolo / seguito da 2.
Si possono fare due osservazioni: l'asterisco non puo' essere omesso, come si usa fare in algebra; il calcolatore esegue alla lettera cio' che incontra, senza potersi inventare alcunche'. Se avessimo scritto C1C2 (omettendo cioe' l'operatore *), tale scritta sarebbe interpretata come un unico identificatore di variabile numerica; dato poi che per il computer solo i primi due caratteri sono significativi, la variabile A verrebbe ad essere definita come il rapporto di C1 per 2.
La seconda osservazione e' che non e' necessario chiudere il prodotto di C1 per C2 tra parentesi: il risultato viene ugualmente corretto (riguarda in proposito le considerazioni svolte sulle precedenze nei calcoli, lezione 4, pagina 7, paragrafo 5).

LINEA 50 - In questa riga l'istruzione PRINT ordina la stampa della dicitura posta tra virgolette, seguita dal valore numerico depositato nella variabile numerica A. Il punto e virgola fa in modo che tale numero sia stampato di seguito a quanto compare tra le virgolette. Ricorda che ogni numero e' preceduto dal segno (omesso se e' positivo) ed e' seguito da uno spazio. Il segno = posto tra le virgolette non e' un operatore aritmetico, ma fa parte di una stringa. Il valore di A visualizzato e' quello calcolato alla linea precedente.

LINEA 60 - Qui c'e' poco da spiegare, oramai: vengono lasciate tre righe video. Come per la linea 30, siamo in presenza di istruzioni multiple, separate le une dalle altre dai due punti.

LINEA 70 - In quest'ultima riga si usa l'istruzione GOTO (lezione 2, pagina 8, paragrafo 8). Il controllo del programma passa alla linea 20 con un salto incondizionato; risulta evidente che, riciclando il procedimento visto fin qui (ad eccezione del CLS iniziale), non si esce piu' dal programma; per farlo occorre dare un break.

Per finire, ancora qualche considerazione. La linea 40 risulta comoda per chi deve leggere ed interpretare il programma, ma l'uso della variabile A non e' necessario; infatti si puo' eliminare completamente la riga 40 e scrivere la 50 nel modo seguente:

```
50 PRINT "AREA=" ; C1*C2/2
```

Potremmo poi attaccare il contenuto della linea 30 in coda alla 20, trasformandola in una riga ad istruzioni multiple; in modo del tutto analogo, puo' essere eliminata anche la linea 60, portando il suo contenuto alla fine della 50:

```
(10)    10 CLS
        20 INPUT "CATETI" ; C1,C2 : PRINT : PRINT
        30 PRINT "AREA=" ; C1*C2/2 : PRINT : PRINT : PRINT
        40 GOTO 20
```

Ci sembra che il listato sia ancora facilmente leggibile; il miglioramento rispetto alla versione precedente consiste in un minor ingombro di memoria ed in una maggiore velocita' in esecuzione. E' comunque difficile apprezzare quest'ultimo fattore, dal momento che il computer compie in ogni caso un numero ridottissimo di operazioni; lo stesso procedimento, applicato ad un programma di piu' ampie proporzioni, accorcerebbe i tempi d'esecuzione in maniera piu' evidente.

Il programma (10) gira piu' veloce del (9) per due motivi: e' stata eliminata un'istruzione (definizione della variabile numerica A), ed altre sono passate in coda a linee preesistenti: il computer fa prima ad eseguirle, in questo modo.
Infatti un programma viene eseguito nel modo seguente: il calcolatore prende la linea con l'etichetta piu' bassa, la traduce in linguaggio macchina (ossia trasforma le varie istruzioni del BASIC nei codici numerici equivalenti, che sono gli unici che in effetti puo' eseguire), e la mette in atto. Successivamente prende la label immediatamente superiore, la traduce e la esegue, e cosi' via fino a che non arriva alla fine del programma.

Si suol dire, in termini informatici, che il BASIC e' un linguaggio INTERPRETATO linea per linea. E' quindi intuitivo che il computer faccia prima ad interpretare ed eseguire un minor numero di linee, se non altro per il tempo che risparmia nella ricerca dell'etichetta seguente.

A titolo informativo, sappi che esistono anche i COMPILATORI, ossia dei programmi particolari che trasformano un programma in BASIC nell'equivalente scritto in linguaggio macchina: non riga per riga, ma tutto intero. Il risultato ottenuto e' il programma compilato, che, messo in esecuzione, risulta molto piu' veloce di quello interpretato, in quanto non esistono piu' tutte le traduzioni di ogni linea, una per volta. Il compilatore non esegue il programma, ma lo traduce soltanto in linguaggio macchina.

Eccoti una versione 'supercompressa' del programma precedente:

(11)
```
10 CLS
20 INPUT"CATETI";C1,C2:PRINT:PRINT:PRINT"AREA="C1*C2/2:PRINT:PRINT:
   PRINT:GOTO20
```

Esso esegue esattamente le funzioni dei programmi (9) e (10), in un tempo minore ed occupando meno memoria; risulta pero' di difficile lettura. Inoltre, nel caso volessimo aggiungere delle istruzioni, bisogna andare a cercare il giusto punto di innesto; in alcuni casi potrebbe anche essere necessario riscriverne una parte. Avrai notato che non e' stato possibile condensare tutto in un'unica linea: l'istruzione CLS verrebbe eseguita ad ogni riciclo, cancellando il valore dell'area non appena e' stato scritto, impedendone cosi' la lettura. Dato che CLS serve solo al primo giro del programma, per cancellare tutte le scritte preesistenti sul monitor, la linea 10 potrebbe anche essere eliminata, senza con cio' pregiudicare il corretto funzionamento di quello che resta.

Oltre ad aver eliminato tutti gli spazi superflui, osserva che e' stato tolto anche il punto e virgola che segue la dicitura AREA=. La cosa e' ammessa ed il programma gira regolarmente.
Questa e' una caratteristica del BASIC di NUOVA ELETTRONICA, e non vale per tutti i computer che usano BASIC diversi.

Il punto e virgola che e' associato all'istruzione INPUT non puo' invece essere omesso; in caso contrario si avrebbe una segnalazione d'errore.

Crediamo di aver esaminato abbastanza a fondo il programma proposto. Naturalmente in futuro non potremo soffermarci cosi' a lungo su ogni aspetto della programmazione. Oltretutto rischieremmo di essere troppo prolissi e di annoiarti. L'intendimento seguito e' stato quello di comunicarti un certo metodo logico di affrontare la programmazione in BASIC. Sara' poi la tua esperienza personale a suggerirti, volta per volta, il migliore modo di procedere.

Per quanto ci riguarda, ci fa piacere pensare di essere riusciti a trasmetterti il convincimento che la base prima su cui costruire un programma ben fatto sia l'ordine mentale e materiale. La pratica diretta, e solo essa, potra' insegnarteli veramente.

La lezione 10 e' terminata. Dopo la stampa del riepilogo puoi uscire dal programma ed eseguire quello che hai scritto nell'esercitazione. Prova poi a scrivere anche le altre due versioni proposte, per renderti conto del loro funzionamento.

RIEPILOGO DELLA LEZIONE 10

SCRITTURA DI UN PROGRAMMA

E' buona norma numerare le linee di 10 in 10. le etichette possono essere ottenute in via automatica col comando AUTO. e' consigliabile fare uso degli spazi, anche se non sono indispensabili, per rendere piu' chiaro e leggibile il listato. Solo in casi particolari (poca memoria disponibile, oppure ricerca di un'alta velocita' di esecuzione) conviene contravvenire a questo metodo.

INPUT

Con l'istruzione INPUT possono essere introitati i valori di piu' variabili.

```
Es.     630 INPUT A1,A2,A3,A4
Es.     410 INPUT "introduci sette numeri interi" ; N1%,N2%,N3%,N4%,N5%,N6%,N7%
```

L E Z I O N E n. 11

INDICE

1 - MEM - (da MEMORY=MEMORIA)

Il comando diretto MEM serve per conoscere il numero di celle di memoria ancora disponibili. Il suo impiego tipico e' il seguente:

```
(1)       PRINT MEM
```

Esso va impartito a livello comandi, ossia quando e' presente la dicitura READY. MEM puo' anche essere usato a mo' d'istruzione, ossia e' possibile inserirlo all'interno di un programma, come nell'esempio che segue:

```
(2)       715 IF MEM<50 THEN PRINT "RESTA POCA MEMORIA" : GOTO 300
```

In pratica, quindi, MEM e' una variabile numerica speciale riservata al sistema.

Puo' succedere che un programma lasci libere pochissime locazioni di memoria; in questo caso, pur essendo contenuto tutto in macchina, potrebbe anche non girare. Questo comportamento si spiega tenendo presente che il computer usa le celle di memoria ancora libere per depositarvi i valori delle variabili usate. E' quindi intuibile che il programma potrebbe anche non girare, proprio per mancanza di memoria.

Ci sembra il caso di ricordare che la memoria di un computer viene misurata prendendo come unita' di confronto il cosidetto 'cappa', cosi' definito:

```
(3)       1K = 1024 LOCAZIONI DI MEMORIA
```

Il valore 1024 non e' casuale: si tratta della potenza indice 10 del numero 2. Quando si dice, ad esempio, che un computer ha 40K di memoria, non si deve pensare che si abbiano a disposizione 40000 celle di memoria; il loro numero esatto e' 1024*40=40960.

Se si impartisce l'ordine di caricare un programma in memoria centrale prelevandolo da disco, puo' succedere che la memoria disponibile non sia sufficiente per contenerlo tutto; in un caso come questo si ottiene la seguente segnalazione d'errore:

```
(4)       OUT OF MEMORY          (oltre i limiti di memoria)
```

Ovviamente il programma non viene caricato.

Cio' non significa che un programma molto lungo non possa in alcun modo essere eseguito con quella disponibilita' di memoria: bisogna spezzarlo in due o piu' parti, concatenandole tra di loro col comando RUN messo alla fine di ogni programma parziale. Questa procedura talvolta e' molto semplice, mentre in altre occasioni pone dei grossi problemi per trasportare i valori delle variabili da un programma a quello che gli e' concatenato. Vedremo piu' avanti come ci si comporta in simili frangenti.

Queste lezioni di BASIC in autoistruzione costituiscono un esempio pratico del meccanismo di concatenamento di programmi: infatti ti sarai reso conto che ogni lezione e' formata da due o tre parti agganciate tra di loro. Questo e' stato

fatto proprio per permettere l'esecuzione di ogni lezione anche a coloro che dispongono di soli 40K di memoria.

Riprendendo il discorso di spezzare un programma in due o piu' parti per riuscire a caricarlo in memoria, c'e' da dire che tale operazione va fatta in sede di stesura del medesimo: se non si riesce a caricare un programma perche' risulta troppo lungo, non esiste nessun artificio, a quel punto, per aggirare l'ostacolo. Il concatenamento deve essere realizzato da chi elabora il programma, e non puo' essere messo in atto a posteriori, se non prendendo il listato originario e riscrivendoselo tutto, per poterlo cosi' spezzare in diverse parti.

2 - CLEAR - (LIBERA)

Quando si accende il computer e si caricano DOS e BASIC, il sistema mette automaticamente a disposizione dell'utente 50 locazioni di memoria ad uso delle stringhe. Se questo numero di caratteri non e' sufficiente, e le stringhe elaborate da un programma ne occupano un numero maggiore, si ottiene la seguente segnalazione d'errore:

(5) OUT OF STRING SPACE (superato lo spazio delle stringhe)

Esiste ovviamente il modo di allargare la disponibilita' di celle di memoria destinate alle stringhe. Se, ad esempio, per il corretto funzionamento di un programma servissero 350 caratteri di stringa, all'inizio del medesimo occorre inserire la seguente istruzione:

(6)
```
.....
30 CLEAR 350
.....
```

E' abbastanza intuitivo che lo spazio riservato alle stringhe va a sottrarsi a quello disponibile per il programma: la somma di queste due quantita' non puo' superare l'intera disponibilita' di memoria, altrimenti si ottiene la segnalazione d'errore (4), cui si aggiunge il numero di linea in cui si verifica l'inconveniente.

L'istruzione CLEAR, pero', assolve anche ad un'altra funzione: azzera tutte le variabili, siano esse numeriche o di stringa. Quindi la riga 30 dell'esempio appena visto non solo riserva lo spazio specificato per le stringhe, ma mette a zero ogni variabile numerica presente in memoria a quel momento; analogamente, tutte le variabili di stringa vengono annullate. Occorre quindi fare molta attenzione alla posizione di programma dove si inserisce CLEAR; normalmente si tratta delle primissime linee, ma puo' capitare di usarlo anche in una zona intermedia, proprio per azzerare i valori di tutte le variabili elaborate fino a quel momento. Ecco un esempio di applicazione:

(7)
```
.....
1840 CLEAR
.....
```

Facciamo notare, a questo proposito, due comportamenti importanti: un'istruzione

CLEAR data nel modo visto ora non altera la disponibilita' di caratteri riservati alle stringhe, ossia non cambia quanto stabilito da una dichiarazione del tipo (5). La seconda osservazione e' che un'istruzione come la (8) effettua una chiusura automatica degli eventuali file che risultano aperti a quel momento. Dato che non abbiamo ancora parlato di file e del loro trattamento, torneremo su questo aspetto di CLEAR al momento piu' opportuno.

3 - FRE - (da FREE=LIBERO)

Sotto certi aspetti il comando FRE e' un doppione di MEM: infatti fornisce il numero di celle di memoria ancora disponibili.
Il modo d'uso e' pero' diverso: FRE deve essere seguito da un parametro racchiuso tra parentesi, altrimenti si ottiene una segnalazione d'errore. Questo parametro puo' essere numerico o di stringa, ed il risultato fornito dal calcolatore e' diverso nei due casi.

Esaminiamo dapprima il caso che il parametro sia numerico; se scriviamo, a livello di comandi diretti:

```
(8)       PRINT FRE(0)
```

abbiamo un comportamento identico a MEM: otteniamo il numero di locazioni di memoria ancora libere. Il numero messo tra parentesi, nel nostro esempio, e' zero, ma nulla vieta di metterne uno diverso:

```
(9)       ? FRE(3.45)
          ? FRE(1000)
          PRINT FRE(A)
```

Qualsiasi valore numerico, compreso quello di una qualunque variabile numerica (anche se non precedentemente inizializzata), fa compiere la medesima funzione. Rispetto a MEM c'e' solo lo svantaggio di dover fare un maggior numero di battute, quindi in tale applicazione non riveste molta importanza.

Se invece il parametro messo tra parentesi e' di tipo stringa, il computer stampa ancora un numero, ma esso non rappresenta piu' la disonibilita' residua di memoria, bensi' il numero di caratteri ancora a disposizione delle stringhe. Tale discorso e' quindi collegato con quanto detto poc'anzi a proposito di CLEAR. Anche la stringa messa tra parentesi puo' essere di tipo qualunque, come puoi vedere dagli esempi che seguono:

```
(10)      PRINT FRE(B1$)
          PRINT FRE("prova")
          PRINT FRE("ELISA")
          PRINT FRE("..")
          PRINT FRE("  ")
          PRINT FRE("")
```

Tutti questi esempi, se eseguiti nelle stesse condizioni, forniscono un unico risultato. L'ultima possibilita' ci sembra quella piu' pratica da usare.

Finora abbiamo visto esempi di FRE usato come comando diretto; esso, analogamente a MEM, puo' essere impiegato anche come istruzione all'interno di una linea di programmazione; facendo un esempio simile al (2), fatto per MEM, si potrebbe scrivere:

```
(11)      120 IF FRE("")<1 THEN PRINT "NON C'E' SPAZIO PER LE STRINGHE"
```

Il comando/istruzione FRE puo' quindi trovare utili applicazioni in diverse occasioni.

ESERCIZIO n. 1

In base a quanto e' stato detto a proposito di MEM e FRE, ritieni possibile che i seguenti comandi:

```
PRINT MEM          PRINT FRE("")
```

diano due numeri, col secondo maggiore del primo?

.....
.....
.....
.....

RISPOSTA

L'evento proposto e' possibile; basti pensare che, qualora il primo numero fosse maggiore del secondo, si potrebbe riservare alle stringhe un numero piu' alto di caratteri (col comando CLEAR), in modo da abbassare il primo ed aumentare il secondo: si puo' quindi fare in modo che FRE("") fornisca un numero maggiore di quello dato da MEM (o da FRE(0)).

ESERCIZIO n. 2

Supponiamo che il comando PRINT MEM fornisca il numero 750; e' possibile, a questo punto, impartire il comando CLEAR 1000?

.....
.....
.....
.....

RISPOSTA

Il numero 750 rappresenta il totale di celle di memoria disponibili a quel dato momento; non e' quindi possibile riservarne 1000 alle stringhe, ma al massimo, appunto, 750. Tutto nell'ipotesi, sottintesa, di non aver dato in precedenza una

CLEAR per un numero di caratteri uguale o superiore a 250: in tale caso, infatti, mancherebbero 750 locazioni (o meno) per arrivare a 1000, quindi la cosa sarebbe possibile.

ESERCIZIO n. 3

Con quale comando e' possibile ridurre a zero la disponibilita' di caratteri riservati alle stringhe?

.....
.....
.....
.....

RISPOSTA

Il comando (o l'istruzione, se usata in una linea di programma) e' il seguente:

CLEARO

In questo modo, ovviamente, non si deve fare uso di stringhe; in compenso si liberano le 50 locazioni di memoria che sono automaticamente riservate ad esse.

---

4 - IF...THEN...ELSE - (SE...ALLORA...ALTRIMENTI)

Cominciamo ora ad esaminare l'istruzione composita IF...THEN...ELSE. Essa, al pari del ciclo FOR...NEXT, e' uno dei pilastri del linguaggio BASIC. Le cose da dire al riguardo sono molte, ed e' preferibile esporle poco per volta, per mezzo di esempi pratici. Per questo motivo non esauriremo l'argomento nella lezione presente.

Abbiamo gia' visto, molto fugacemente, un'applicazione di questa istruzione (lezione 9, pagina 7, programma (10)), quindi dovresti avere un'idea, seppure sommaria, del suo funzionamento.

Per esaminare piu' a fondo il funzionamento di IF...THEN (vedremo piu' avanti l'estensione ad ELSE), riprendiamo il programma (3) della lezione 9, a pagina 4, che per comodita' riportiamo nuovamente:

(12)
```
10 FOR C=1 TO 13 STEP 2
20 PRINT C;
30 NEXT C
```

Come certamente ricorderai, si tratta di un loop che stampa sul monitor i sette numeri dispari che vanno da 1 a 13.
Ci proponiamo, ora, di scrivere un programma che esegua le stesse cose, utiliz-

zando pero' le istruzioni IF...THEN al posto di FOR...NEXT:

```
(13)      10 N=1
          20 PRINT N;
          30 N=N+2
          40 IF N<=13 THEN GOTO 20
```

Facciamone una lettura ragionata:

LINEA 10 - Per cominciare, la variabile numerica N viene inizializzata al valore uno; essa e' usata per depositarvi i numeri dispari da stampare, e cio' spiega il perche' della scelta fatta;
LINEA 20 - Qui ordiniamo la stampa sul monitor del valore della variabile N. Al primo giro, evidentemente, otterremo il numero 1, primo dei numeri dispari richiesti;
LINEA 30 - Ora la variabile N viene definita uguale al valore precedente aumentato di due: ad ogni giro, e vedremo che ne verranno fatti diversi, essa viene incrementata di due, fornendo cosi' via via i successivi numeri dispari cercati;
LINEA 40 - Questa riga dice: se N e' minore od uguale a 13, allora vai alla linea 20. E' ovvio quindi che si ottiene, alla 20, la stampa del numero 3; siamo cosi' al secondo passaggio per quella linea. Si tratta quindi di un procedimento ricorsivo, che viene ripetuto con valori crescenti di N, fino a che non si arriva al numero 13, che viene stampato. Giunti a quel punto, la riga 30 incrementa l'ultimo valore di N portandolo a 15; in tal caso la condizione posta dalla linea 40 non e' piu' verificata, quindi il programma non salta piu' alla riga 20, ma prosegue, ossia si arriva alla fine; sul monitor compare la scritta READY che indica il ritorno al livello di comandi diretti. In precedenza e' stata stampata la sequenza dei numeri dispari da 1 a 13, uno di seguito all'altro per la presenza del punto e virgola nella linea 20.

Quando la condizione posta alla linea 40 risulta verificata, si ha un SALTO CONDIZIONATO alla riga 20; questo e' un concetto estremamente importante, in quanto l'istruzione GOTO, da sola, rappresenta invece un SALTO INCONDIZIONATO, che avviene cioe' sempre, senza sottostare ad alcuna condizione.
Bisogna notare la grande somiglianza concettuale tra i due programmi appena visti. Nel primo di essi la variabile C viene usata per il controllo di un loop che comincia dal valore 1 e deve arrivare a 13 con incrementi di 2; nella seconda linea si stampa il valore corrente di C, che viene incrementato e riciclato alla riga seguente. Tutte queste considerazioni valgono pari pari anche per il programma (13), dove la variabile N ha preso il posto di C. La similitudine e' ancor piu' evidente se scriviamo il secondo programma in questo modo:

```
(14)      10 N=1
          20 PRINT N;
          30 N=N+2 : IF N<=13 THEN 20
```

In pratica la terza e quarta riga sono state unite, ottenendo una somiglianza formale maggiore col programma (12).
E' evidente che in effetti il computer compie operazioni diverse nelle varie li-

nee dei due programmi: infatti la riga 30 del secondo incrementa la variabile e ricicla il procedimento se essa e' minore o uguale a 13; invece la riga 30 del programma (12) incrementa la varabile di controllo e fa tornare alla linea 10; e' qui che avviene il riciclo, fino a che la variabile non oltrepassa il valore finale 13. Il controllo esercitato dall'istruzione IF...THEN alla fine del suo programma viene messo in atto, nell'altro, all'inizio.

L'analogia e' forse ancor piu' evidente in questa nuova versione:

```
(15)      10 N=1
          20 IF N>13 GOTO 40 ELSE PRINT N;
          30 N=N+2 : GOTO 20
          40 END
```

Ora la linea 40 e' necessaria, ma si puo' pensare che esista anche nella versione (12). Con queste varianti i programmi (12) e (15) sono pressoche' identici, pur facendo uso di istruzioni completamente diverse.

Ci preme fare notare queste considerazioni per mostrare come lo stesso problema possa essere risolto in modi diversi, seguendo procedimenti differenti che portano comunque ai medesimi risultati. In generale, si puo' dire che viene applicato piu' spesso il ciclo FOR...NEXT: lo schema mentale che esso presuppone e' assimilato velocemente ed ogni programmatore si abitua a farne un largo uso.
Bisogna solo fare attenzione al fatto che il loop potrebbe venire percorso un numero di volte inaspettato, a seconda dei valori assunti dai parametri che lo controllano. Tale inconveniente non si ha con cicli governati dall'istruzione IF...THEN, il cui comportamento e' piu' trasparente.

In generale, possiamo dire che ogni ciclo FOR...NEXT puo' essere sostituito da un'opportuna istruzione IF...THEN, mentre in molte applicazioni di quest'ultima non e' possibile pensare la sostituzione con FOR...NEXT. L'una o l'altra, quindi, vanno usate con criterio, cercandi di sfruttare al massimo le loro potenzialita' intrinsèche.

Se andiamo a confrontare il contenuto dei tre programmi (13), (14) e (15), notiamo innanzitutto che in una linea di programma e' indifferenre scrivere:

```
(16)      IF N<=13 THEN GOTO 20
          IF N<=13 THEN 20
          IF N<=13 GOTO 20
```

Ognuna di queste istruzioni risulta corretta; quella di mezzo raccoglie le nostre preferenze, ed e' quella piu' comunemente usata.
La linea 20 del programma (15) fornisce un primo esempio applicativo dell'istruzione ELSE (=altrimenti). Essa non puo' essere usata da sola, ma deve seguire le precedenti IF...THEN; queste ultime due, invece, possono fare a meno di ELSE.

Puoi immaginare facilmente che mettendo assieme i vari operatori di confronto:

```
(17)      <    =    >    <=    >=    <>    NOT   OR   AND
```

(i primi sono matematici, gli ultimi tre sono logici), si possono ottenere infi-

nite possibilita' operative. E' facile, usando opportunamente i vari parametri, gestire con poche righe di programma anche dei fenomeni estremamente complessi.

Quelli visti sono i primi cenni sull'uso dell'istruzione IF...THEN...Else. Dovresti comunque aver chiaro quello che e' uno dei comportamenti piu' importanti: se l'ipotesi avanzata da IF non e' verificata, il controllo del programma passa alle istruzioni che seguono ELSE; se esso manca, si passa alla linea seguente. Fa' attenzione a questo comportamento: spesso un programma non gira come ci si aspetterebbe perche' non si e' tenuto conto di questa regola. Se una IF non e' verificata, il computer ignora tutto quello che segue e salta alla riga successiva. Se, ad esempio, si scrive:

(18)
```
350 ...
360 IF A=2 THEN 100 : GOTO 430
370 ...
```

si commette sicuramente un errore: infatti se A vale 2 il programma va alla linea 100, mentre passa alla 370 in caso contrario; l'istruzione GOTO 430 non puo' quindi essere raggiunta in nessun caso.

La lezione 11 e' terminata. Dopo la stampa del riepilogo puoi esercitarti a fare qualche piccola applicazione di IF...THEN. Potresti poi studiare piu' a fondo il funzionamento del programma (11) riportato a pagina 7 della lezione 9; molto probabilmente non ti era del tutto chiaro, mentre ora potresti capirlo bene. Si tratta di un metodo classico per ordinare numeri in modo crescente; se riesci a padroneggiarlo, potresti provare ad apportarvi alcune modifiche, in modo che con esso si possano ordinare valori numerici in ordine decrescente. Riprenderemo, in futuro, l'argomento.

RIEPILOGO DELLA LEZIONE 11

MEM

Il comando/istruzione MEM e', in pratica, una variabile speciale riservata al sistema; in essa si trova depositato il numero di locazioni di memoria ancora libere. Dopo aver caricato il DOS ed il BASIC, si puo' conoscere questo dato col comando:

```
PRINT MEM
```

Il numero ottenuto esprime la quantita' di memoria lasciata libera dal sistema operativo e dal BASIC; tieni presente che 50 caratteri vengono riservati automaticamente alle stringhe. La disponibilita' totale effettiva va quindi aumentata di 50.

CLEAR

Col comando/istruzione CLEAR si possono svolgere due funzioni: CLEAR dato da solo azzera tutte le variabili (numeriche e di stringa). Se invece CLEAR e' seguito da un numero, oltre al risultato precedente si ottiene anche quello di riservare alle stringhe un numero di caratteri uguale a quello indicato.

```
Es.     CLEAR
Es.     290 CLEAR 255
Es.     10 CLEAR 0
```

FRE

Il comando/istruzione FRE deve essere seguito da un parametro messo tra parentesi; se esso e' numerico (qualsiasi), si ottiene un funzionamento uguale a quello di MEM. Se invece si mette tra parentesi una stringa qualsiasi, si ha un numero che corrisponde alla disponibilita' di caratteri per le stringhe.

```
Es.     ?FRE(13)
Es.     PRINT FRE (X$)
```

IF...THEN...ELSE

Con queste istruzioni si eseguono dei test matematici o logici; a seconda che essi siano piu' o meno verificati, si possono far seguire al programma strade diverse. Con queste istruzioni e' anche possibile eseguire dei cicli ripetitivi, come se si trattasse di un loop FOR...NEXT.

```
Es.     IF B>A THEN PRINT "troppo piccolo"
Es.     IF D=34 THEN D=E-7
```

LEZIONE n. 12

INDICE

1 - STRINGHE -

Questa lezione e' dedicata alle stringhe; per cominciare, definiremo il concetto in modo esauriente, poi cominceremo ad esaminare in dettaglio alcune delle parole chiave previste per il loro trattamento (16 in tutto).
Ti sarai gia' reso conto che non si puo' parlare di programmazione in BASIC senza esaminare le stringhe. Anche questo argomento, quindi, riveste una grande importanza: e' praticamente impossibile trovare un programma che non faccia uso di stringhe, siano esse sotto forma di variabili o poste tra virgolette.
Devi sapere, inoltre, che un tipo di file (quello RANDOM) tratta esclusivamente dati espressi sotto forma di stringhe. Occorre quindi acquisire una buona familiarita' nel loro trattamento.

L'appendice 1 della prefazione riporta, tra le parole chiave, anche quelle che si riferiscono alle stringhe; per comodita' le trascriviamo qui sotto, riunite tutte assieme:

(1)

1 - ASC	9 - MID$
2 - CHR$	10- MKD$
3 - CVD	11- MKI$
4 - CVI	12- MKS$
5 - CVS	13- RIGHT$
6 - INSTR	14- STR$
7 - LEFT$	15- STRING$
8 - LEN	16- VAL

Le istruzioni 3-4-5-10-11-12 sono quelle preposte al trattamento dei numeri nei file random, e le vedremo al momento piu' opportuno.

Cominciamo col definire con esattezza il concetto di stringa: con questa parola si intende un insieme di lettere, numeri o altri caratteri a piacere. Ecco qualche esempio a caso:

(2)
```
NUOVA ELETTRONICA
Peso massimo sopportato (in kg.)
CLIENTE #
*** SCADENZE DI FINE MESE ***
29 marzo 1974
56.7 + 0.82
S/N (scegliere)
```

Come vedi, qualsiasi raggruppamento di caratteri puo' essere definito come una stringa. In particolare, le lettere possono essere sia maiuscole che minuscole. Una stringa puo' contenere anche soltanto dei numeri: ricorda pero' che il computer li tratta come caratteri e non come cifre. Il penultimo esempio ne e' una prova; facendo stampare quella stringa non si ottiene il valore della somma!

Innanzitutto bisogna sapere che la lunghezza massima di una stringa e' di 255 caratteri; se si cerca di farne una piu' lunga si ottiene la seguente segnalazione d'errore:

(3)
```
STRING TOO LONG          (stringa troppo lunga)
```

Bisogna poi tenere presente quanto e' stato detto nella lezione precedente al paragrafo 2 (istruzione CLEAR, pagina 4): il sistema riserva in via automatica 50 caratteri alle stringhe.

2 - CHR$ - (da CHARACTER=CARATTERE)

A pagina 9 dell'introduzione l'appendice 3 riporta una tabella; in essa compaiono 96 elementi, numerati da 32 a 127: ad ogni numero corrisponde il carattere scritto a fianco. Si vede, ad esempio, che al numero 71 si trova la lettera G maiuscola; il medesimo carattere, in minuscolo, porta il numero di codice 103.
La tabella porta l'intestazione CODICI ASCII; cosa significa? I computer, come sai, sono nati negli Stati Uniti, e ben presto si e' sentita l'esigenza di standardizzare il loro funzionamento. A dire il vero, non e' che si siano fatti grandi passi in quella direzione: praticamente ogni calcolatore elettronico ha un suo modo specifico di funzionare; e' cosa normale che anche due macchine fabbricate dallo stesso costruttore abbiano comportamenti diversi.

I codici ASCII (American Standard Code for Information Interchange = codice standard americano per lo scambio di informazioni), in ogni computer a 8 bit, vanno da 1 a 255. Per i vari computer, normalmente, si ha una perfetta corrispondenza per i codici che vanno dal 32 al 126, anche se dal 91 al 96 e dal 123 in poi si possono avere alcune differenze.

L'istruzione CHR$ e' quella che da' la corrispondenza tra i codici ASCII ed i vari caratteri trattati dal computer. Tale asserzione e' vera con le limitazioni appena dette.

CHR$ puo' essere usato anche a livello di comandi diretti; ecco un esempio:

```
(4)       PRINT CHR$(65)
```

Si vede che CHR$ e' seguito da un numero posto tra parentesi: al variare di quel parametro si ottengono, via via, i diversi caratteri trattati dal computer. Con l'esempio appena visto si ha la stampa della lettera A (maiuscola).
Il parametro posto tra parentesi puo' essere dato anche sotto forma di variabile numerica o di espressione algebrica:

```
(5)       PRINT CHR$(B)
          PRINT CHR$(2*53-(7+FF)/C)
```

A questo riguardo c'e' da dire che il parametro tra parentesi, nel caso fosse negativo, viene arrotondato all'intero inferiore:

```
(6)       PRINT CHR$(68)
          PRINT CHR$(68.12)
          PRINT CHR$(68.93)
```

Questi tre comandi forniscono sempre la lettera D, in quanto il parametro tra parentesi viene arrotondato a 68.

Anche ai codici numerici inferiori a 32 o superiori a 127 corrispondono dei ca-

ratteri o delle funzioni particolari (come il ritorno del cursore in alto a sinistra, oppure la stampa di un carattere grafico); il discorso si allargherebbe troppo, quindi torneremo sull'argomento piu' avanti. Per ora ci preme piuttosto insistere sul concetto di carattere associato ad un numero di codice; una considerazione importante, ad esempio, e' la seguente: non e' detto che ci sia concordanza tra cio' che viene visualizzato dal monitor e quello che viene scritto dalla stampante. Se quest'ultima, ad esempio, e' una EPSON, al codice 123 fa corrispondere una parentesi graffa aperta, mentre il monitor mostra una parentesi quadra. Altro esempio: al codice 94 corrisponde una freccia verso l'alto (ossia il simbolo dell'elevamento a potenza, nel computer); anche nella EPSON si ha il medesimo risultato (la freccia manca pero' dell'asta ed assomiglia piu' ad un accento circonflesso); in una stampante MICROLINE si ottiene invece una freccia orientata verso sinistra. C'e' quindi, a tutt'oggi, una certa confusione tra i costruttori, cosa che ingenera imbarazzo o confusione nell'utente finale.
Da quanto detto si puo' dedurre che l'appendice 3 dell'introduzione da', in effetti, i codici della stampante usata e non quelli del computer: questi ultimi sono quelli visualizzati sul monitor. Per rendersi conto personalmente e direttamente del reale comportamento di monitor e stampante, nei riguardi di un certo numero di codice, basta scrivere, a livello di comandi diretti, quanto segue (l'esempio e' stato fatto per il codice 96, ma puo' essere ripetuto per un numero a piacere):

```
(7)       PRINT CHR$(96) : LPRINT CHR$(96)
```

Sul monitor compare il carattere che il calcolatore associa al numero 96, mentre la stampante esegue lo stesso compito: i risultati possono essere uguali o diversi, a seconda della stampante usata.

ESERCIZIO n. 1

Questo esercizio non comporta, da parte tua, una risposta; sul monitor compare una tabella coi codici ASCII da 91 a 127. I caratteri ad essi associati valgono per il computer, mentre, come abbiamo gia' detto, l'appendice n.3 dell'introduzione vale per la stampante. Confronta quindi queste due tabelle, per trarne le dovute conclusioni. Una differenza la noterai senza dubbio: l'alfabeto del monitor e' visualizzato maiuscolo, mentre per la stampante esso e' minuscolo.
Sono stati scelti i limiti anzidetti perche' per quelli dal 32 al 90 non vi sono differenze tra computer e stampante.

Fino ad ora abbiamo visto applicazioni di CHR$ come comando diretto; trattandosi pero' di una vera e propria istruzione, essa puo' essere impiegata all'interno di un programma.
Vedremo un'applicazione pratica di CHR$ alla fine di questa lezione.

Per concludere questo primo incontro coi codici ASCII, facciamo notare che il carattere associato al codice 32 e' quello dello spazio, cosa che non e' evidente nell'appendice n.3 dell'introduzione.

Nel caso vengano omesse le parentesi, si ha la seguente segnalazione d'errore:

```
(8)       SYNTAX ERROR          (errore di sintassi)
```

Se il parametro racchiuso tra parentesi e' fuori dal campo 1-255 si ha:

```
(9)       ILLEGAL FUNCTION CALL          (uso illecito della funzione)
```

L'istruzione CHR$(0) non fornisce alcun risultato, e non da' errore.

3 - ASC - (da ASCII)

L'istruzione ASC svolge una funzione inversa a quella di CHR$: infatti fornisce il numero di codice ASCII associato ad un certo carattere. Ecco un esempio:

```
(10)      PRINT ASC("M")
```

Anche questa applicazione vede l'istruzione ASC impiegata come comando diretto; si ottiene il numero 77, ossia il codice ASCII del carattere M (si faccia sempre riferimento all'appendice n.3 dell'introduzione).

Si puo' quindi dire che anche l'istruzione ASC richiede un parametro tra parentesi. Esso deve essere di tipo stringa; evidentemente, allora, puo' essere dato sia tra virgolette che sotto forma di variabile alfanumerica:

```
(11)      PRINT ASC("*")
          A$="*" : PRINT ASC(A$)
```

I due esempi forniscono lo stesso codice: 42.
Vediamo un'altra possibilita':

```
(12)      PRINT ASC("elefante")
```

Come risultato si ottiene il numero 101, ossia il codice ASCII della lettera E minuscola: se ne deduce che la funzione ASC tratta solo il primo carattere della stringa posta tra parentesi ed ignora i successivi.

Omettendo le parentesi si ha una segnalazione d'errore uguale alla (8) vista poco fa per CHR$.
L'istruzione seguente causa una segnalazione d'errore uguale alla (9):

```
(13)      PRINT ASC("")
```

ESERCIZIO n. 2

Osserva il seguente comando:

```
          PRINT ASC(CHR$(57))
```

Quale risultato si ottiene?

. . . . . .
. . . . . .
. . . . . .
. . . . . .

RISPOSTA

Si ottiene il numero 57, dal momento che si chiede il numero di codice del carattere che ha per codice 57. Il risultato si spiega anche pensando che ASC e CHR$ sono due funzioni inverse l'una dell'altra.

Quello che segue e' un programma estremamente semplice, che fa uso dell'istruzione ASC:

(14)

```
10 CLS
20 INPUT A$
30 IF ASC(A$))47 AND ASC(A$)(58 THEN PRINT "cifra" : GOTO 20
40 IF ASC(A$))64 AND ASC(A$)(91 THEN PRINT "lettera maiuscola" :
   GOTO 20
50 IF ASC(A$))96 AND ASC(A$)(123 THEN PRINT "lettera minuscola" :
   GOTO 20
60 PRINT "ne' cifra ne' lettera" : GOTO 20
```

Vediamolo in dettaglio, riga per riga:

LINEA 10 - Viene cancellato lo schermo.
LINEA 20 - Il computer resta in attesa di una stringa, introdotta da tastiera con la pressione del tasto RETURN. Tale stringa puo' essere formata da diversi caratteri, ma tanto vale fermarsi al primo, dal momento che l'istruzione ASC che segue alle righe seguenti agisce solo sul primo carattere. La stringa viene associata alla variabile alfanumerica A$.
LINEA 30 - Dall'appendice n.3 si deduce che le cifre da zero a nove sono comprese tra i codici ASCII 48 e 57. Questa riga di programma ordina al computer di stampare la dicitura "cifra" se il codice del primo carattere di A$ e' compreso nel campo citato. Se tale condizione e' verificata, si ha poi il passaggio alla linea 20, ossia si riparte da capo. In tal modo, allora, il calcolatore ha riconosciuto che la stringa digitata inizia con un numero.
LINEA 40 - Il programma arriva a questa riga solo se la condizione posta a quella precedente non e' verificata. qui si ha un comportamento analogo a quello visto per la linea 30: se il codice ASCII del primo carattere di A$ e' compreso tra 65 e 90, si ottiene la dicitura "lettera maiuscola" e si torna alla riga 20.
LINEA 50 - Se le condizioni precedenti non sono soddisfatte, si arriva alla riga 50; essa e' in grado di riconoscere se il primo carattere di A$ e' una lettera minuscola, con un procedimento del tutto simile ai precedenti. In caso di risposta affermativa si ha la dicitura "lettera minuscola" e si torna alla riga 20.

LINEA 60 - Il programma arriva a questo punto solo se le tre precedenti condizioni non sono verificate; cio' significa che il primo carattere di A$ e' diverso da una cifra o da una lettera (maiuscola o minuscola). Viene quindi stampata la frase tra virgolette, e si torna alla riga 20, da cui ricomincia il ciclo.

Il programma non ha mai termine; per uscirne occorre fare un break.
L'istruzione AND (=E) impone che le due condizioni siano verificate contemporaneamente; se una sola di esse e' soddisfatta , la IF non e' realizzata e si va alla riga seguente.

Ci sono alcuni caratteri che mandano in errore il programma: virgolette, virgola, due punti, line feed, RETURN, spazio. Con alcuni di questi l'errore avviene sempre, mentre con altri cio' accade solo all'inizio del programma, quando si introduce la stringa A$ per la prima volta.
Questo comportamento e' dovuto al modo di funzionare dell'istruzione INPUT, come vedremo meglio parlando di LINEINPUT. In pratica, coi caratteri prima menzionati, si ottiene per A$ una stringa vuota, e si ha la segnalazione d'errore (9).

In qualcuna delle prossime lezioni vedremo che e' possibile esercitare un controllo sugli errori, per fare in modo da non uscire mai da un programma quando essi si verificano.

4 - LEN - (da LENGTH=LUNGHEZZA)

Come lascia intendere la parola chiave, LEN fornisce la lunghezza della stringa specificata. Anche questa funzione richiede l'uso delle parentesi e, al pari di CHR$ e ASC, puo' essere usata anche come comando diretto.
Ecco un primo esempio:

```
(15)      G$="LEZIONI DI BASIC" : PRINT LEN(G$)
```

Digitando quanto sopra a livello di comandi diretti, dopo la pressione del tasto RETURN si legge sul monitor il numero 16: tanti sono i caratteri compresi tra le due virgolette; nel conto vengono messi pure gli spazi, anche se fossero all'inizio o alla fine della stringa.
Riportiamo qualche esempio; tra parentesi compare il numero restituito dalla funzione LEN:

```
(16)      PRINT LEN("computer")                                  (8)
          PRINT LEN(" computer ")                                (10)
          AW$="" : PRINT LEN(AW$)                                (0)
          K$="33.005" : PRINT LEN(K$)                            (6)
          A$="Computer" : B$=" NE-Z80" : PRINT LEN(A$+B$)        (15)
          CC$(4)="-29+3*5 (kg.)" : PRINT LEN(CC$(4))             (13)
```

I primi due casi differiscono per gli spazi che precedono e seguono la parola posta tra virgolette. Il terzo esempio mostra che l'istruzione LEN funziona anche con stringhe di lunghezza nulla (stringhe vuote).
Il quinto caso fa vedere che l'argomento di LEN puo' essere costituito dalla somma di due o piu' stringhe; tale operazione e' accettata dal computer, mentre

ogni altra lo manda in errore.
Nell'ultimo esempio si puo' vedere che anche le variabili alfanumeriche possono essere indicizzate; nel caso specifico, CC$(4) e' il quarto elemento di un vettore. Nota l'uso delle parentesi nell'istruzione LEN: il numero delle parentesi aperte deve essere uguale a quello delle chiuse; se si scrivesse LEN(CC$(4) il calcolatore andrebbe in errore (SYNTAX ERROR).

Il parametro da mettere tra parentesi deve essere di tipo stringa, altrimenti si ha una segnalazione d'errore. Ad esempio, col comando:

```
(17)      LPRINT LEN(45)
```

si otterrebe la seguente dicitura:

```
(18)      TYPE MISMATCH          (carattere non adatto)
```

La lezione e' finita. Dopo la stampa del riepilogo, troverai una esercitazione eseguita solo sul monitor. Si tratta di un programma che mostra i caratteri corrispondenti ai codici ASCII da 1 a 255. Probabilmente resterai sorpreso dall'effetto prodotto da alcuni codici, specie per quelli inferiori a 32; cerca di renderti conto del loro funzionamento. In futuro, comunque, riprenderemo l'argomento per approfondirlo.
Il programma e' ciclico, ossia riparte da capo automaticamente; premendo la barra dello spazio puoi fermarlo momentaneamente. Per terminare occorre premere il tasto RETURN; solo cosi' si ha accesso al menu' finale.

RIEPILOGO DELLA LEZIONE 12

STRINGHE

Per stringa si intende un insieme di caratteri alfanumerici. Gli identificatori delle variabili di tipo stringa sono seguiti dal simbolo $; per il resto seguono le stesse regole degli identificatori di variabili numeriche. Anche le variabili di stringa possono essere indicizzate.
La lunghezza massima di una stringa e' di 255 caratteri. Il sistema riserva alle stringhe 50 caratteri; se ne servono di piu', occorre creare il posto necessario con l'istruzione CLEAR.

CHR$

Questa funzione di stringa richiede che sia specificato un parametro di tipo numerico, posto tra parentesi. Si tratta di una variabile speciale di tipo alfanumerico, riservata al sistema; in essa viene depositato il carattere corrispondente al codice ASCII richiesto. Il parametro puo' essere dato sotto forma di numero, di variabile numerica e di espressione algebrica.

ASC

ASC e' una variabile numerica speciale riservata al sistema; vi viene depositato un numero che da' il codice ASCII della stringa messa in argomento tra le parentesi. Svolge una funzione inversa a quella di CHR$.

LEN

Anche LEN e' una variabile numerica speciale, in cui viene depositata la lunghezza della stringa specificata in argomento.

L E Z I O N E n. 13

INDICE

1 - LEFT$ - (LEFT=SINISTRA)

Anche questa lezione e' dedicata al trattamento delle stringhe, come quella precedente. Iniziamo con l'istruzione LEFT$, dandone subito un esempio applicativo:

```
(1)       R$=LEFT$("RADIOAMATORE",5)
```

Con questa definizione, la variabile alfanumerica R$ contiene la parola RADIO; infatti la (1) dice che R$ e' uguale ai primi cinque caratteri (a sinistra) della stringa posta tra virgolette.
Si vede quindi che la funzione LEFT$ necessita di due parametri, posti tra parentesi: il primo indica la stringa di partenza, il secondo dice quanti caratteri prendere alla sua sinistra; si puo' anche dire che il primo parametro deve essere di tipo stringa, mentre il secondo e' numerico.
Al solito, al posto di una stringa tra virgolette si puo' mettere qualsiasi altra espressione, come pure per il parametro numerico.
Ecco qualche esempio significativo:

```
(2)       PRINT LEFT$("COMPUTER",3) + LEFT$("PITONE",4)
          A$="FABBRICATO" : B$=LEFT$(A$,6)
          C$="BISOGNO" : K=3 : D$=LEFT$(A$,K)+C$
          E$="CANTERANO" : F$=LEFT$(A$,2*K)+LEFT$(E$,K+2)
```

Nel primo caso si hanno due istruzioni LEFT$: con una si forma la dicitura COM, in quanto si prendono le prime tre lettere di COMPUTER; con l'altra si prelevano i 4 caratteri a sinistra di PITONE, ossia PITO. La parola risultante dalla somma dei due motti e' COMPITO.
Il secondo esempio fornisce per B$ la parola FABBRI (6 caratteri a sinistra della stringa A$).
Nel terzo caso, partendo dal precedente A$ e definendo C$ e K come indicato, si ottiene in D$ la parola FABBISOGNO (somma dei primi 3 caratteri di FABBRICATO con l'intero C$).
Nell'ultimo esempio la stringa F$ e' definita come somma di sei caratteri a sinistra di A$ (FABBRI) coi primi cinque caratteri di E$ (CANTE): si ha quindi la parola FABBRICANTE.

Ovviamente quelli visti possono sembrare solo dei giochetti fini a se stessi; essi servono per farti capire il meccanismo di funzionamento dell'istruzione LEFT$. Gli effettivi casi pratici daranno un senso piu' completo e reale al discorso.

Se si fa qualche errore nel dare l'istruzione LEFT$, si ottiene una segnalazione d'errore che puo' essere di uno dei seguenti tipi:

```
(3)       SYNTAX ERROR
          ILLEGAL FUNCTION CALL
          TIPE MISMATCH
```

Omettendo uno dei parametri, o entrambi, o le parentesi, ecc., si ha errore di sintassi. Se il numero che specifica i caratteri da prelevare e' minore di zero o maggiore di 255 si ottiene la seconda dicitura. L'ultima, ad esempio, si ottiene invertendo l'ordine del parametro stringa e di quello numerico.

Le possibilita' d'errore sono molteplici; quelle viste servono d'esempio. Da notare che i due esempi che seguono non danno luogo ad errore:

```
(4)     A$="PROVA" : PRINT LEFT$(A$,0)
        PRINT LEFT$(A$,21)
```

Nel primo caso si ha una stringa nulla; nel secondo si ottiene la parola PROVA, e detta stringa e' lunga 5 caratteri e non 21 come richiesto.

2 - RIGHT$ - (RIGHT=DESTRA)

L'istruzione RIGHT$ ha un funzionamento perfettamente equivalente a quello visto per LEFT$, tranne che essa lavora, ovviamente, alla destra della stringa indicata. Tutte le considerazioni svolte poco fa per LEFT$ possono essere trasportate a RIGHT$.

Vediamo un esempio di utilizzazione:

```
(5)     10 CLS
        20 PRINT "introdurre una parola" : INPUT X$
        30 PRINT
        40 UL$=RIGHT$(X$,1)
        50 IF UL$="e" OR UL$="E" OR UL$="i" OR UL$="I" PRINT "--PLURALE"
        60 IF UL$="o" OR UL$="O" OR UL$="a" OR UL$="A" PRINT "--SINGOLARE"
        70 PRINT : PRINT
        80 GOTO 20
```

Esaminiamolo velocemente:

LINEA 10 - Cancella il monitor.
LINEA 20 - Stampa la dicitura tra virgolette, poi visualizza un punto interrogativo nella riga seguente, per la presenza dell'istruzione INPUT.
LINEA 30 - Lascia una riga video.
LINEA 40 - Definisce la stringa UL$ come ultimo carattere a destra della stringa X$ introdotta alla riga 20.
LINEA 50 - Stampa la dicitura '--PLURALE' se UL$ e' uguale alla lettera E oppure I (sia minuscole che maiuscole).
L'istruzione OR (=OPPURE) serve per porre condizioni multiple; nel nostro caso sono quattro. Se una sola di esse e' verificata si ha la stampa della parola '--PLURALE', altrimenti il programma passa alla riga successiva.
LINEA 60 - Il contenuto e' analogo alla riga precedente: si ha la stampa della dicitura '--SINGOLARE' se la stringa UL$ contiene la lettera O oppure A (sia minuscole che maiuscole).
LINEA 70 - Serve per saltare due righe sul monitor.
LINEA 80 - Rimanda il programma alla linea 20: il ciclo si ripete all'infinito.

Facciamo qualche osservazione:

1 - Evidentemente il programma e' fatto in modo da distinguere solo le terminazioni nelle vocali I/E/O/A; ogni altra lettra finale della parola introdotta

non viene riconosciuta e si passa alla linea 20 senza stampare alcunche'. Evidentemente questo costituisce un difetto; del resto il programma vuole essere solo il primo approccio alla soluzione del problema, a scopo esemplificativo.

2 - Difetto ancora maggiore e' costituito dal fatto che ogni parola che termina con la lettera E e' considerata plurale: INDICE, GIUDICE, ecc. non lo sono. Vale quanto detto or ora: si tratta di un semplice esempio, senza troppe pretese; l'importante e' che tu ti renda conto di come l'istruzione RIGHT$ (e quindi anche LEFT$) potrebbe essere impiegata. A titolo di esercizio potresti cimentarti nel tentativo di modificare il programma proposto, in modo che giri correttamente in ogni caso.

3 - La linea 20 potrebbe essere scritta anche nel modo seguente:

```
20 INPUT "introdurre una parola";X$
```

Sul monitor, pero', sarebbe apparso questo messaggio:

```
introdurre una parola?
```

La soluzione adottata, invece, mostra quanto segue:

```
introdurre una parola
?
```

che va decisamente meglio. Il punto interrogativo, come sai, e' introdotto automaticamente dal computer, quando questo incontra l'istruzione INPUT.

4 - Si e' usato l'identificatore UL$ come mnemonico di ULTIMO; il sistema avrebbe accettato anche ULTIMO$, pur prendendo in considerazione solo i primi due caratteri.

5 - Nota l'assenza della parola chiave THEN prima di PRINT (righe 50 e 60); la cosa e' lecita e non causa errore. Il programma gira un po' piu' velocemente e si risparmia qualche cella di memoria.

3 - MID$ - (da MIDDLE=CENTRO)

La funzione MID$ serve per estrarre caratteri nel mezzo di una stringa. Il funzionamento e' quindi simile a quello di LEFT$ e RIGHT$. Anche MID$ necessita di parametri, in numero di tre: il primo indica la stringa da trattare, il secondo serve per individuare la posizione del carattere da cui ha inizio l'estrazione, il terzo specifica quanti caratteri bisogna prelevare.

Vediamo un esempio:

```
(6)       A$="CERBOTTANA" : B$=MID$(A$,4,5)
```

Nella variabile alfanumerica B$ viene depositata la parola BOTTA, ottenuta prendendo 5 caratteri di CERBOTTANA a partire dal quarto.

Le varie segnalazioni di eventuali errori sono sempre quelle riportate in (3). Per i tre parametri valgono le solite considerazioni (virgolette, stringhe, espressioni, ecc.).

Osserva questi due esempi:

```
(7)     C$=MID$(A$,2,0)
        C$=MID$(A$,113,72)
```

Nel primo caso si ottiene sempre una stringa vuota; nel secondo C$ e' piu' o meno lunga (puo' essere anche vuota), a seconda della lunghezza di A$. In questi casi non si hanno segnalazioni d'errore. Invece nell'esempio che segue si ha errore per uso illecito della funzione:

```
(8)     PRINT MID$(A$,0,3)
```

L'istruzione MID$ puo' essere impiegata in un modo molto interessante, come mostra questo esempio:

```
(9)     P$="SUORA" : MID$(P$,4,1)="L" : PRINT P$
```

la stampa fornisce la parola SUOLA. In pratica, quindi, la funzione MID$ e' stata usata per cambiare un carattere all'interno di una stringa.
Ecco qualche altra elaborazione della stringa P$ precedente:

```
(10)    MID$(P$,1,1)="N"                    (NUORA)
        MID$(P$,2,4)="ACCO"                 (SACCO)
        MID$(P$,2,8)="ACCOCCIA"             (SACCO)
        MID$(P$,4,2)="I "                   (SUOI )
```

Entro le parentesi sono riportati i contenuti di P$, caso per caso: notare che il secondo e terzo caso forniscono lo stesso risultato, e che P$, nell'ultimo esempio, e' lungo sempre 5 caratteri (quello finale e' uno spazio).

Questa particolarita' e' esclusiva di MID$; provando ad applicarla anche con LEFT$ e RIGHT$ si ottiene una segnalazione d'errore (SYNTAX ERROR).
Resta da dire che l'istruzione MID$, opportunamente impiegata, puo' sostituire LEFT$ e RIGHT$. Vediamo un esempio:

```
(11)    A$="TREMARE" : B$=RIGHT$(A$,4) : C$=MID$(A$,4,4)
```

Nelle stringhe B$ e C$ va a finire la medesima parola: MARE.

Passiamo ora all'esecuzione di alcuni esercizi; in essi devi applicare le funzioni di stringa appena viste. Fa' molta attenzione al testo dei problemi e segui fedelmente le istruzioni che troverai scritte sul monitor.
Probabilmente troverai qualche difficolta' nell'eseguire gli esercizi proposti; cerca di fare del tuo meglio e non scoraggiarti in caso di insuccesso. E' molto importante, naturalmente, che tu capisca alla perfezione le spiegazioni fornite per ogni problema.

ESERCIZIO n. 1

Con quale istruzione si puo' trasformare la stringa T$ (contenente la parola

EQUESTRE) in una stringa T$ uguale a PEDESTRE?
.....
.....
.....
.....

RISPOSTA

Si deve scrivere:

```
MID$(T$,1,3)="PED"
```

Con questa istruzione i primi tre caratteri di T$ sono trasformati in PED, e si ottiene il risultato desiderato. Ovviamente anche l'istruzione:

```
T$="PEDESTRE"
```

raggiunge lo stesso scopo.
Dopo aver eseguito l'istruzione indicata, se si chiede la stampa della variabile alfanumerica T$ col comando:

```
PRINT T$
```

si ottiene appunto la parola PEDESTRE. E' chiaro che tutte queste considerazioni hanno valore solo se la variabile T$ era stata definita in precedenza.

ESERCIZIO n. 2

Supponiamo che la variabile di stringa L$ sia cosi' definita:

```
L$="CONTRORDINE"
```

Quale istruzione si deve dare per trasformarla in "CON ORDINE" (con due spazi intermedi)?
.....
.....
.....
.....

RISPOSTA

Si deve usare sempre la funzione MID$, nel modo seguente:

```
MID$(L$,4,2)="  "
```

All'interno delle virgolette vanno messi due spazi.

ESERCIZIO n. 3

Abbiamo tre stringhe definite in questo modo:

```
A1$="NUOVAMENTE"
A2$="ELETTRONI"
A3$="VERONICA"
```

Scrivi un'istruzione che sia in grado di depositare nella variabile NE$ la scritta:

NUOVA ELETTRONICA

Devi sfruttare le variabili gia' definite.
.....
.....
.....
.....

RISPOSTA

Il miglior modo per definire NE$ come richiesto e' il seguente:

```
NE$=LEFT$(A1$,5)+" "+A2$+RIGHT$(A3$,2)
```

Infatti si prendono i primi 5 caratteri di A1$, si somma uno spazio, poi si prende A2$ per intero, ed infine si aggiungono gli ultimi due caratteri di A3$.

Per ora terminiamo l'esame delle istruzioni che servono al trattamento delle stringhe; in una delle prossime lezioni esamineremo le funzioni INSTR, STR$, STRING$ e VAL; ad esclusione delle 6 funzioni CV ed MK, che vedremo quando parleremo della gestione dei file random, avremo passato in rassegna tutte le funzioni di stringa.

4 - REVON - (da REVERSE ON=NEGATIVO INSERITO)

Il computer e' in grado di presentare anche caratteri scritti in negativo. Nella scrittura normale i caratteri sono chiari su fondo scuro, mentre in quella in negativo i caratteri sono scuri in campo chiaro.
L'istruzione REVON serve appunto per passare dalla scrittura normale a quella in reverse (=negativo). Come per molte altre istruzioni, anche REVON puo' essere impiegata come comando diretto.
Per fare un esempio concreto, riprendiamo il programma (5) visto a pagina 2, ed aggiungiamo questa linea di programma:

```
(11)      35 REVON
```

Dopo averla incontrata, il computer sa che tutte le istruzioni di stampa seguen-

ti devono avvenire in negativo: le diciture "--PLURALE" e "--SINGOLARE" avverranno quindi in quel modo.

La riga aggiunta poteva essere inserita anche nella posizione 25, con risultati uguali: ogni istruzione PRINT (ossia ogni riga di monitor saltata) non tiene conto di REVON, che agisce solo quando si stampa effettivamente qualcosa (un numero, una stringa, una variabile, ecc.).

5 - REVOFF - (da REVERSE OFF=NEGATIVO DISINSERITO)

Dopo aver impartito l'istruzione REVON, tutte le operazioni di stampa avverrebbero in negativo; per poter tornare alla scrittura normale bisogna impartire l'istruzione REVOFF, che esplica quindi la funzione inversa di REVON.

Torniamo al programma (5); dopo l'introduzione della linea 35, ogni stampa avviene in negativo, mentre il nostro obiettivo e' quello di avere in reverse solo le diciture delle linee 50 e 60. Per raggiungere lo scopo basta scrivere la seguente riga di programma:

```
(12)      65 REVOFF
```

Come nel caso della 35 vista prima, anche REVOFF puo' essere inserito in una posizione diversa, ad esempio la 75.
In pratica, allora, prima di tornare alla linea 20 bisogna impartire l'istruzione REVOFF; in tal modo la stampa della frase che compare nella 20 avviene in positivo.

NOTA - Nella prima versione del BASIC le istruzioni REVON e REVOFF non devono essere seguite da spazi vuoti; nelle righe ad istruzioni multiple, qualora una di quelle istruzioni sia seguita dai due punti, occorre avere l'avvertenza di non mettere uno spazio tra l'istruzione stessa ed il carattere dei due punti. Se lo si facesse si otterrebbe una segnalazione d'errore. Tale inconveniente e' stato eliminato nelle versioni successive del BASIC.

Col funzionamento in reverse i codici ASCII subiscono delle variazioni. alla fine della lezione c'e' una esercitazione che ti mostra la corrispondenza tra numeri di codice e caratteri sia quando ci si trova in condizioni normali di scrittura che in negativo. In pratica succede che ad un certo numero di codice e' associato un carattere in positivo ed un carattere diverso in negativo; sara' l'esercizio a farti capire tale concetto, meglio di tante parole.

La lezione e' finita. Dopo il riepilogo troverai l'esercitazione menzionata poco fa. Sul monitor verranno scritte le relative istruzioni.

RIEPILOGO DELLA LEZIONE 13

LEFT$

Questa funzione tratta la parte sinistra della stringa specificata come primo parametro tra le parentesi; il numero dei caratteri interessati e' indicato dal secondo parametro.

RIGHT$

Questa funzione e' analoga alla precedente, con la differenza che tratta la parte destra della stringa specificata.

MID$

L'istruzione MID$ serve per avere accesso ad una parte intermedia della stringa specificata tra i parametri. Questi debbono essere tre: la stringa su cui operare, un numero che da' la posizione del carattere di partenza all'interno della stringa, ed un altro numero che specifica quanti caratteri si vogliono considerare.

REVON

L'istruzione REVON innesca la scrittura in negativo.

REVOFF

Questa istruzione annulla l'effetto di REVON, ossia fa tornare alla scrittura normale in positivo.

L E Z I O N E n. 14

INDICE

1 - PRINT TAB - (da PRINT WITH TABULATION=STAMPA CON TABULAZIONE)

Abbiamo gia' parlato dell'istruzione PRINT (lezione 2, pagina 4), spiegando il modo piu' semplice di impiegarla. Esistono pero' anche altre applicazioni, come avrai constatato osservando l'elenco delle parole chiave (APPENDICE n.1 dell'introduzione, pagine 6 e 7).

Ecco l'elenco delle varianti permesse da PRINT:

```
(1)     PRINT
        PRINT #
        PRINT @
        PRINT TAB
        PRINT USING
```

Ognuna di esse puo' essere abbreviata, in sede di scrittura di programma, usando il carattere del punto interrogativo (?) al posto della parola chiave PRINT.
La seconda variante riguarda il trattamento dei file random, quindi la vedremo a tempo debito. Le altre istruzioni hanno invece attinenza diretta con la visualizzazione dei dati sul monitor, e poco per volta le vedremo tutte. In questa lezione vedremo l'impiego di PRINT TAB.

Ogni riga video e' composta da 32 caratteri; il primo porta il numero di tabulazione zero, il secondo 1, il terzo 2, e cosi' via fino a 31 (ultimo carattere della riga).
L'istruzione PRINT TAB (che puo' essere scritta anche PRINTTAB, oppure ? TAB, od anche ?TAB) serve per visualizzare i dati ad iniziare dalla tabulazione specificata dal parametro numerico posto tra parentesi:

```
(2)     PRINT TAB(8)"TABULAZIONE 8"
```

Questo comando fa stampare la dicitura tra virgolette ad iniziare dal nono carattere della riga interessata alla stampa.

Come abbiamo gia' visto in altri casi, il parametro numerico puo' essere dato sotto forma diretta, come nell'esempio appena visto, oppure come variabile numerica o espressione algebrica. Vediamo qualche applicazione corretta:

```
(3)     1  PRINT TAB(3)"ora"
        2  ? TAB (25) 25
        3  PRINTTAB(C7)"seconda mensilita'"
        4  ?TAB(2*A!)200
        5  PRINTTAB(10)"VALORE:";TAB(20);T
        6  PRINTTAB(2);IN;"-";TAB(9);"integrati LSI"
        7  PRINTTAB(2)"TABULAZIONE 2"
        8  ?TAB(34);-59,3
        9  ?TAB(28)1000
        10 ?TAB(28)"TABULAZIONE 28"
```

Per far vedere come vengono visualizzati i vari esempi fatti, devi fare riferimento anche alla mappa video, riportata nell'APPENDICE n.4 dell'introduzione, a pagina 10. La prima linea di quella figura rispecchia quanto detto in precedenza

sulla numerazione delle tabulazioni all'interno di ogni riga video.
Nella figura che segue rappresenteremo ogni carattere non coinvolto dalla stampa con un punto; ogni riga avra' sempre 32 caratteri, alcuni dei quali sono punti (quelli non interessati dalla stampa dei dati). I numeri a sinistra richiamano gli esempi appena visti; di fianco ad essi, tra parentesi, sono scritti i valori assegnati alle variabili impiegate nei vari casi.
Tenendo presenti queste convenzioni, ecco come verrebbero visualizzati, uno per uno, i diversi esempi:

```
1                    ...ora.............................

2                    ...............................25....

3  (C7=4)            .....seconda mensilita'............

4  (A!=6)            ..............200..................

5  (T=-2714)         ...........VALORE:...-2714........

6  (IN=733)          ...733.-.integrati LSI............

7                    ..TABULAZIONE 2...................

8                    ..................................

                     ..-59.............3................

9                    ..................................

                     .1000.............................

10                   ................................TABU

                     LAZIONE 28........................
```

Vediamoli in dettaglio.
1 - Abbiamo detto che il primo carattere ha tabulazione zero; la tabulazione 3, quindi, fa iniziare la stampa dal quarto carattere.
2 - PRINT e' stato abbreviato col punto interrogativo; il numero 25 e' stampato a cominciare dal 27.esimo carattere di riga (26 di tabulazione piu' uno per il segno). Tutti gli spazi messi nell'esempio possono essere eliminati, in quanto il computer li ignora: ?TAB(25)25 darebbe il medesimo risultato.
Anche ?TAB(25);25 fornisce una stampa identica. Come vedi, la flessibilita' d'uso dell'istruzione PRINT e' davvero notevole.
3 - Ammesso che C7 assuma il valore 4 (come e' indicato tra parentesi), si ha la visualizzazione indicata.
4 - Per A!=6 la tabulazione vale 12: il parametro puo' essere specificato sotto forma di espressione algebrica. Nel caso presente si e' fatto uso della variabile A! in semplice precisione.
Il numero 200 inizia al 14.esimo carattere; valgono le considerazioni fatte poco fa per il punto e virgola prima di 200.
5 - Questo esempio mostra che PRINTTAB puo' essere ripetuto piu' volte; bisogna

pero' tenere presente che ogni TAB fa riferimento all'inizio della riga video interessata. I due punti e virgola possono essere omessi; e' comunque consigliabile non abusare di questa possibilita', in quanto il listato risulta meno leggibile; inoltre non tutti i BASIC godono di questa facilitazione.
Puo' accadere che la seconda tabulazione abbia un valore tale che il secondo dato dovrebbe essere sovrapposto al primo, come nell'esempio che segue:

```
PRINTTAB(10)"VALORE:";TAB(15);T
```

In casi come questo il computer non va in errore, ma effettua la stampa del secondo dato immediatamente di seguito al primo:

```
..........VALORE:-2714..........
```

6 - In questo esempio non c'e' niente di nuovo; il numero 733 (valore della variabile numerica IN) comincia al quarto carattere.
La dicitura 'integrati LSI' verrebbe automaticamente spostata verso destra qualora la lunghezza del numero contenuto da IN fosse superiore a 5 cifre. Ad esempio, col valore 125000 si otterrebbe:

```
...125000.-integrati LSI........
```

Vediamo altre due stampe, corrispondenti ai valori 4 e 2306004 di IN:

```
...4.-...integrati LSI..........

...2.306E+06.-intregrati LSI...
```

Il carattere '-' segue il valore numerico, in quanto e' legato ad esso dal punto e virgola. Nel primo caso la frase parte regolarmente dal decimo carattere. Forse il secondo esempio ti lascia perplesso; se rammenti quanto e' stato detto nelle prime lezioni a proposito di variabili numeriche in semplice e doppia precisione, ti renderai conto che il numero 2306004 supera la lunghezza di 6 cifre, e per questo motivo viene rappresentato con la notazione in virgola mobile (nota che si perde l'ultima cifra: il numero trattato dal computer diventa 2306000; per evitare questa approssimazione bisogna usare al posto della variabile in semplice precisione IN quella in doppia precisione IN#).

7 - Questo esempio, accostato al precedente, rende evidente il fatto che la visualizzazione di un dato numerico riserva un caratere iniziale al segno.

8 - Con questa istruzione si dice al computer di stampare il numero -59 alla tabulazione 34; viene quindi lasciata una linea vuota, ed il numero in questione parte col segno meno dal terzo carattere della riga seguente (infatti esso e' il 35.esimo carattere, se si comincia a contare dall'inizio della riga precedente).
L'istruzione 8 riporta, dopo il numero -59, una virgola e poi il numero 3. Sappiamo gia' che cio' non significa che si vuole la stampa del numero decimale -59,3 (che e' tale solo nella notazione italiana); la virgola causa un salto al diciassettesimo carattere della riga video, come abbiamo spiegato alle pagine 4 e 5 della lezione 2. Il numero 3 e' alla 18.esima posizione, per la solita considerazione sul segno.

9 - In questo caso succede che il dato numerico da stampare non riesce a stare per intero nella riga; allora il computer ignora la tabulazione e lo stampa alla riga seguente. Da notare che se avessimo chiesto la stampa di un numero formato da meno di 4 cifre, essa avrebbe rispettato la tabulazione; in queste considerazioni entra in ballo anche il carattere vuoto che pone fine ad ogni numero.

10- Il comportamento appena descritto vale solo per la stampa dei numeri; essi non vengono mai spezzati, a differenza delle stringhe. Quest'ultimo esempio lo dimostra chiaramente: la stringa tra virgolette parte dalla tabulazione stabilita, e viene scritta in parte nella riga di partenza, nella parte rimanente in quella che segue.

Le cose da dire sarebbero molteplici; con questa breve rassegna abbiamo comunque esaminato quelli che sono i comportamenti piu' importanti. Mano a mano che si presentera' l'occasione, illustreremo altre particolarita'.

Per concludere, esaminiamo i possibili errori. Se si fanno sbagli di battitura o si usano le parentesi in modo scorretto, si ha questa segnalazione:

(4) SYNTAX ERROR

Se il parametro numerico tra parentesi assume valori negativi o superiori a 255 si ha la seguente dicitura:

(5) ILLEGAL FUNCTION CALL

Infine, se si usa un parametro alfanumerico al posto di quello numerico, si ottiene questo messaggio:

(6) TYPE MISMATCH

Si tratta di segnalazioni che abbiamo gia' incontrato perche' rispecchiano comportamenti simili a quelli che possono accadere con altre istruzioni. La storia si ripete, anche coi computer!

Vediamo ora alcuni esercizi. In essi dovrai applicare le nozioni svolte finora, adottando un po' di buon senso per le situazioni che presentano qualche elemento di novita'.

Proseguendo nell'apprendimento della programmazione, ti accorgerai che ogni programma presenta sempre alcune considerazioni nuove, che vanno risolte caso per caso; prima si affronta il problema dal lato teorico, poi si stende il programma risultante e lo si fa girare. Se tutto va bene, tanto meglio; piu' spesso pero' occorre apportare delle modifiche, perche' non si erano previsti certi comportamenti del computer in determinate situazioni. Si apportano allora le modifiche opportune, e cosi' via fino ad arrivare alla soluzione finale corretta.

In pratica, ogni programma comporta questo lavoro di approssimazioni successive; esiste addirittura un termine apposito per definire questo modo d'operare: DEBUG o DEBUGGING. Esso deriva dalla parola BUG, che in inglese significa PULCE; si tratta quindi di SPULCIARE il programma, togliendo tutti quegli inghippi che ne impediscono un corretto funzionamento.

Esiste anche un detto, che contiene una grande verita': THERE IS ALWAYS ONE MORE BUG (c'e' sempre qualche altra pulce). Se opererai con questa filosofia, non stancandoti mai di provare, modificare e riprovare, affinerai velocemente le tue capacita' di programmazione. Il trucco sta tutto qui. Certamente questo comporta una buona dose di pazienza e, perche' no?, di cocciutaggine. Imparare cose nuove implica sempre sacrificio, ma la soddisfazione che si puo' conseguire ripaga ampiamente delle fatiche.

L'importante e' non demordere mai.

ESERCIZIO n. 1

Usando le convenzioni adottate finora (coi punti, riga di 32 caratteri), scrivi il risultato fornito dalla seguente istruzione di stampa:

```
PRINT TAB(G+F/2);5000
```

supponendo che le variabili abbiano questi valori: G=7; F=5.

```
..........
..........
..........
..........
```

RISPOSTA

La tabulazione e' data sotto forma di espressione algebrica; il suo valore e' 7+5/2=9.5; ovviamente il computer non puo' tabulare su una frazione di carattere, quindi arrotonda all'intero inferiore. Il parametro vale quindi 9, e la riga ottenuta ha questa configurazione:

```
.........5000.................
```

Il numero 5000 comincia dal decimo carattere per la solita considerazione sul segno.

ESERCIZIO n. 2

Supponiamo di avere la seguente istruzione di stampa:

```
PRINTTAB(4)-12;TAB(2)"PROVA"
```

scrivi il risultato della stampa rispettando le convenzioni precedenti e digi-

tando PROVA in caratteri maiuscoli.
.....
.....
.....
.....

RISPOSTA

La prima parte dell'istruzione fa stampare il numero -12 a cominciare dal quinto carattere della riga video. La tabulazione 2 della parola PROVA non puo' essere rispettata; la dicitura viene quindi visualizzata subito dopo al -12, lasciando pero' un carattere, che e' quello in coda al numero. Si ottiene pertanto:

....-12.PROVA.....

---

2 - INKEY$ - (da INPUT FROM KEYBOARD=RICEVI DALLA TASTIERA)

L'istruzione INKEY$ e' abbastanza singolare. Quando il computer la incontra si mette in cominicazione, per un attimo, con la tastiera; se in quell'attimo viene premuto un tasto, esso viene introitato e puo' essere elaborato dal programma.
In caso contrario il programma prosegue ugualmente.
Detto cosi' puo' sembrare astruso o di nessuna utilita', ma basta l'esempio seguente per dimostrare il contrario:

(7)
```
.....
60 PRINT "PER CONTINUARE PREMERE UN TASTO"
70 X$=INKEY$ : IF X$="" THEN 70
.....
```

Vediamo cosa succede quando il computer arriva alla linea 70 (il contenuto della 60 e' ovvio: si ottiene la stampa della frase tra virgolette). l'istruzione dice: "metti nella variabile X$ il carattere che arriva dalla tastiera in questo momento; se X$ e' vuota allora torna alla linea 70".
A cosa serve questo marchingegno? E' molto semplice: arrivato alla linea 70, se non viene premuto alcun tasto lo svolgimento del programma si arresta, fino a che la condizione richiesta non e' verificata; solo la pressione di un tasto qualunque sblocca la situazione e fa andare il computer alla linea successiva.

Ogni volta che devi separare una pagina, durante lo svolgimento di queste lezioni, il programma presenta appunto un'istruzione di questo tipo.

La grande utilita' della funzione INKEY$ consiste nel fatto che il carattere viene assunto dal computer in via automatica, senza dover premere il tasto RETURN; il suo difetto, se cosi' lo vogliamo chiamare, e' che puo' essere introitato un solo carattere. Pero' si puo' rimediare a questo inconveniente mettendo diverse istruzioni INKEY$ concatenate una dopo l'altra, fino ad arrivare al numero di caratteri richiesti. Vedremo un'applicazione di questo al momento piu'

opportuno; per ora e' meglio non correre troppo, ed approfondire ulteriormente quanto detto fin qui.

La linea 70 dell'esempio (7) puo' anche essere scritta cosi':

```
(8)       70 IF INKEY$="" THEN 70
```

Il programma, in questo caso, gira perfettamente. Le due soluzioni non sono affatto equivalenti in altre occasioni: sono quelle in cui il carattere digitato deve essere trattato dalla parte di programma che segue. Vediamo un esempio:

```
(9)       .....
          470 PRINT "scegliere prego (S/N)"
          480 S$=INKEY$ : IF S$="" THEN 480
          490 IF S$="S" THEN 1230
          500 IF S$="N" THEN 910
          510 GOTO 480
          .....
```

La linea 470 invita l'operatore a scegliere tra due alternative: S (ossia SI) o N (NO); la riga 480 mette in comunicazione il computer con la tastiera. Fino a che non viene premuto alcun tasto, il programma rimane fermo a quel punto. Allorche' viene digitato un carattere qualsiasi, si passa alle linee successive. Se il carattere premuto e' quello della lettera S si salta alla riga 1230; se invece si e' premuto il tasto della N si va alla linea 910. Qualora nessuna di queste due condizioni sia verificata, il controllo del programma passa nuovamente alla riga 480. In pratica quindi il computer avanza solo se viene premuto il tasto S o il tasto N. La cosa non sarebbe possibile senza depositare il carattere digitato in una variabile.

Ecco una variante dell'esempio precedente:

```
(10)      .....
          470 PRINT "scegliere prego (S/N)"
          480 S$=INKEY$ : IF S$<>"S" AND S$<>"N" THEN 480
          490 IF S$="S" THEN 1230 ELSE 910
          .....
```

La linea 480 deposita dapprima in S$ l'eventuale carattere digitato; nella sua seconda parte recita cosi': "se S$ e' diverso da 'S' e diverso da 'N', va alla riga 480". Questa condizione rifiuta quindi sia tutti i caratteri diversi da quei due, che, diciamo cosi', la battuta nulla. Solo se viene premuto il tasto S o N si passa alla linea successiva. In essa si ha un'istruzione IF...THEN, che esegue le stesse funzioni delle righe 490 e 500 del programma (9).
Si tratta, indubbiamente, di un metodo piu' brillante per risolvere il problema.

La linea 480 puo' essere ulteriormente perfezionata nel modo seguente:

```
(11)  480 S$=INKEY$ : IF S$<>"S" AND S$<>"s" AND S$<>"N" AND S$<>"n" THEN 480
```

Cosi' facendo i tasti delle lettere S ed N sono operanti sia in maiuscolo che in minuscolo.

Riassumendo, si puo' dire questo: la battuta causata dall'istruzione INKEY$ deve essere depositata in una variabile per poterla elaborare successivamente. Solo nel caso in cui il carattere digitato non debba essere piu' esaminato si puo' usare INKEY$ in modo diretto.

La lezione e' terminata. Abbiamo trattato due sole istruzioni, che rivestono pero' un'importanza particolare nel BASIC. E' quindi opportuno che tu ti eserciti in modo approfondito nel loro uso pratico. Gli esempi che trovi su queste pagine costituiscono una buona base di partenza, ma non sono sufficienti.

Un buon metodo e' anche quello di prendere in esame un programma gia' fatto e di cercare di interpretarlo correttamente, per comprendere le intenzioni dell'autore. Le riviste ed i libri sono ormai pieni di listati di programmi, quindi il materiale per fare questo tipo di esercizi non ti manca.

Qualora la fonte facesse riferimento a versioni di BASIC un po' diverse (ogni macchina ha una sua versione personalizzata), cimentati nella conversione al tuo computer: e' uno degli esercizi piu' istruttivi.

RIEPILOGO DELLA LEZIONE 14

PRINT TAB

Questa istruzione serve per effettuare visualizzazioni tabulate. Tra parentesi deve essere specificato un parametro numerico (numero, variabile, espressione).

INKEY$

Con questa istruzione il computer si mette in comunicazione con la tastiera ed assume il carattere che risulta premuto in quell'istante. E' utile per inputare un carattere senza dover premere il tasto RETURN.

))

L E Z I O N E n. 15

INDICE

1 - SCRITTURA DI UN PROGRAMMA -

Tra poco dovrai cimentarti nella stesura di un programma, come hai gia' fatto nella lezione 10. Se ricordi, in quell'esercizio il contenuto delle linee da introdurre doveva soddisfare a regole molto rigorose: rispettare gli spazi, scrivere tutto in maiuscolo, eccetera. La cosa era voluta, per costringerti a fare certe operazioni mentali e di tastiera.

Nell'esercitazione che segue, invece, avrai piena liberta' nella digitazione delle varie righe: questo ovviamente vale entro certi limiti. Per ogni linea troverai sempre, sul testo scritto dalla stampante e sul monitor, le istruzioni che ti guideranno passo passo nella scrittura del programma.

Il problema che vogliamo risolvere e' il seguente: immaginiamo di essere in una di quelle ditte che effettuano sondaggi d'opinione, interrogando qua e la' persone scelte a caso; tra i dati richiesti compare anche l'eta' degli intervistati, oltre naturalmente ad altri che per ora non ci interessano.
Supponiamo poi che, per la nostra indagine, l'eta' degli interpellati non debba essere inferiore ad anni 15, mentre verso l'alto non ci sono limiti.

Il programma che ti accingi a scrivere deve essere in grado di introitare le eta' delle varie persone, rifiutandole se risultassero inferiori al valore prefissato. Ci deve poi essere un contatore, che informi costantemente l'operatore circa il numero di dati gia' introdotti.
Vogliamo poi sapere, ad ogni dato digitato, quali sono le eta' minima e massima fino a quel momento, nonche' il valore dell'eta' media tra tutte quelle considerate.

Non devi lasciarti impressionare dall'apparente complessita' delle richieste; la lezione ti condurra' per mano dall'inizio fino alla fine, dove arriverai senza troppo sforzo anche se sei alle prime esperienze di programmazione.

Le istruzioni che dovrai usare sono le seguenti:

```
REM
CLS
INPUT
IF...THEN
GOTO
PRINT
PRINT TAB
REVON
REVOFF
INKEY$
END
```

C'e' quindi gran parte di quello che abbiamo visto finora. Mano a mano che procedi, se hai dei dubbi consulta la spiegazione delle istruzioni alle varie lezioni.

Dovrai inoltre usare alcuni operatori aritmetici e di confronto, nonche' variabili numeriche intere ed in singola precisione e variabili di stringa.

La numerazione delle linee deve cominciare da 100 e procedere di 10 in 10. Se si dovesse digitare direttamente il programma si potrebbe usare il comando:

(1) AUTO 100

Con quello il computer ci fornirebbe automaticamente la numerazione, riga per riga. Quella che ti accingi a fare e' pur sempre una simulazione di stesura di programma, quindi dovrai scriverti da solo le varie etichette.

Il programma si chiama STAT1 (statistica 1); quando avrai terminato di scriverlo te lo ritroverai registrato sul disco nella versione da te effettivamente digitata. Ovviamente, come nel caso della lezione 10, il dischetto non deve avere il nastro di protezione contro le registrazioni, altrimenti il computer non potrebbe incidere il programma.

Ripetiamo ancora che ogni linea viene accettata indipendentemente dal fatto che sia stata digitata in caratteri maiuscoli o minuscoli; inoltre potrai mettere spazi dove ritieni piu' opportuno, tranne che all'interno delle stringhe.
Se non ti riesce di scrivere una riga potrai ottenerla dal computer digitando il carattere del punto interrogativo, come nel programma che hai scritto alla lezione 10.

Per ogni linea scritta ed accettata, il monitor ti mostrera' la versione da noi proposta. Il programma e' formato dalle righe comprese tra le label 100 e 330.
Dopo averle introdotte tutte, troverai il listato del tuo programma confrontato con quello proposto, con tutte le spiegazioni attinenti ad ogni linea.

Adesso possiamo incominciare.

LINEA 100

La prima riga di programma deve contenere l'annotazione:

(2) PROGRAMMA STAT1

Si tratta quindi di una linea di remark, in cui devi fare uso dell'istruzione REM (in sua vece puoi usare anche la relativa abbreviazione).

LINEA 110

Il contenuto di questa linea deve far cancellare lo schermo.

LINEA 120

Questa riga deve servire per chiedere all'operatore l'introduzione dell'eta' delle varie persone intervistate. L'istruzione da usare e' quindi INPUT, segui-

ta dalla dicitura:

(3) ETA'

posta tra virgolette. Il valore dell'eta' deve essere depositato in una variabile numerica intera avente per identificatore la lettera E.

LINEA 130

Se l'eta' introdotta dall'operatore vale zero, si deve passare alla linea 340. Si tratta quindi di un'istruzione del tipo IF...THEN, con cui si deve realizzare un salto condizionato alla linea anzidetta.

LINEA 140

Abbiamo gia' detto che il programma deve rifiutare le eta' inferiori a 15 anni. Questa linea svolge appunto tale funzione. Pertanto occorre fare un controllo su E% con l'istruzione IF...THEN; nel caso che E% risulti minore di 15 deve essere visualizzata la dicitura:

(4) TROPPO BASSA

e si deve tornare alla linea 120. Si tratta quindi di una riga contenente due istruzioni.

LINEA 150

Questa riga deve servire a tenere il conteggio dei dati introdotti. Si usa quindi un contatore N, che in tale linea viene incrementato di uno ad ogni giro. Cio' che scrivi deve valere anche per la prima tornata.

LINEA 160

Indicheremo con MI ed MA i valori minimo e massimo delle diverse eta' via via introdotte. Al primo giro, evidentemente, MI ed MA coincidono con la prima eta' digitata. La linea 160 stabilisce appunto che se N vale 1 le due variabili citate siano inizializzate a quel modo. MI e MA non devono essere di tipo intero.

LINEA 170

Questa riga deve tenere costantemente aggiornato il valore di MI; cio' significa che se il nuovo valore di E% e' inferiore ad MI, allora in MI deve andare a fi-

nire il contenuto di E%. Qui dovrai fare uso dell'istruzione IF...THEN, assieme all'operatore di confronto < (minore).

LINEA 180

Ora devi fare una cosa analoga per la variabile MA; questa volta l'operatore di confronto da impiegare e' > (maggiore).

LINEA 190

Adesso devi introdurre una nuova variabile intera, identificata dalla lettera T, in cui devi accumulare le eta' via via introdotte dall'operatore: ad ogni giro, quindi, quella variabile viene incrementata del nuovo valore di E%.

LINEA 200

Ora bisogna calcolare il valore medio delle eta' introdotte fino ad un certo momento generico. Si tratta quindi di calcolare il rapporto tra T% ed N, depositando tale valore nella variabile intera con identificatore ME.

LINEA 210

In questa linea di programma devi impartire l'ordine di saltare una riga video.

LINEA 220

Qui occorre far stampare sul monitor la frase seguente:

(4) NUMERO DI DATI INTRODOTTI:

Di seguito, poi, deve comparire il valore della variabile N, ossia del contatore che cresce di uno ad ogni giro. Ricorda che il computer trasforma in caratteri maiuscoli tutte le parole chiave e le variabili usate, nel caso vengano digitate in caratteri minuscoli.
Tale conversione non avviene per tutto cio' che viene scritto tra le virgolette, come abbiamo detto piu' di una volta.

LINEA 230

Questa linea deve far saltare una riga video.

LINEA 240

Ora bisogna fare scrivere alla tabulazione 5 la dicitura:

(5) ETA' MINIMA:

Di seguito a questa scritta vogliamo stampare, alla tabulazione 18, il valore della variabile MI.
Ricorda che i valori di tabulazione vanno messi tra parentesi.

LINEA 250

Questa riga di programma fa una cosa analoga alla precedente, riferita pero' all'eta' massima. Alla tabulazione 5 devi quindi far apparire la scritta:

(6) ETA' MASSIMA:

ed aggiungere, alla tabulazione 18, il valore della variabile MA.

LINEA 260

Stesse considerazioni per l'eta' media ME% calcolata alla linea 200. La frase da visualizzare e':

(7) ETA' MEDIA:

Essa va posta alla tabulazione 5; di seguito deve essere stampato, alla tabulazione 18, il valore di ME%.

LINEA 270

Si deve lasciare una riga video.

LINEA 280

Stesso contenuto della linea precedente. Puoi usare anche l'abbreviazione di PRINT, ossia il punto interrogativo.

LINEA 290

Questa riga di programma deve far passare dalla scrittura normale in positivo a quella in negativo.

LINEA 300

Ora devi fare stampare sul monitor la dicitura:

(0) PREMERE UN TASTO

Ricorda che puoi indifferentemente scrivere in caratteri maiuscoli o minuscoli e che puoi usare PRINT o la sua abbreviazione, a tua scelta.

LINEA 310

L'istruzione da scrivere e' quella che fa tornare alla scrittura in positivo, facendo cosi' uscire dal reverse.

LINEA 320

Ora devi fare uso dell'istruzione INKEY$, depositando il valore del carattere digitato nella variabile di stringa con identificatore X. La linea e' ad istruzioni multiple, in quanto subito dopo devi porre la condizione che se X$ e' vuoto, il programma deve tornare all'inizio di questa stessa riga.

LINEA 330

Qui c'e' un salto incondizionato alla riga 110.

LINEA 340

Questa e' l'ultima riga del programma. L'istruzione da mettere e' quella che fa tornare a livello di comandi diretti: il programma, se arriva a questa linea, ha termine.

2 - SPIEGAZIONE DEL PROGRAMMA -

Durante la scrittura del programma il computer, dopo aver controllato il contenuto della riga da te digitata, ti ha mostrato volta per volta una sua possibile configurazione.

Qui di seguito e' riportato il listato dell'intero programma proposto.
Esso segue lo schema di quelli che ti abbiamo presentato finora. Dopo aver seguito la spiegazione del suo funzionamento, lo confronterai con quello che hai

appena finito di scrivere.

(9)

```
100 'PROGRAMMA STAT1
110 CLS
120 INPUT "ETA' ";E%
130 IF E%=0 THEN 340
140 IF E%<15 THEN PRINT "TROPPO BASSA" : GOTO 120
150 N=N+1
160 IF N=1 THEN MI=E% : MA=E%
170 IF E%<MI THEN MI=E%
180 IF E%>MA THEN MA=E%
190 T%=T%+E%
200 ME%=T%/N
210 PRINT
220 PRINT "NUMERO DI DATI INTRODOTTI:";N
230 PRINT
240 PRINT TAB(5) "ETA' MINIMA:";TAB(18);MI
250 PRINT TAB(5) "ETA' MASSIMA:";TAB(18);MA
260 PRINT TAB(5) "ETA' MEDIA:";TAB(18);ME%
270 PRINT
280 PRINT
290 REVON
300 PRINT "PREMERE UN TASTO"
310 REVOFF
320 X$=INKEY$ : IF X$="" THEN 320
330 GOTO 110
340 END
```

Ci soffermeremo solo sulle linee che presentano qualche problema, sorvolando quelle che hanno un contenuto ovvio.

LINEA 120 - Essa serve all'operatore per introdurre le varie eta', che vengono via via depositate nella stessa variabile numerica intera E%; questa assume quindi, ad ogni giro del programma, valori sempre diversi tra di loro. La linea 330 rimanda infatti alla 110, ossia fa ripartire tutto il processo. E' stata usata la variabile intera perche' il dato introdotto puo' essere solo di questo tipo e non supera di certo il valore superiore che limita questo tipo di variabili. Bisogna tenere presente che se l'operatore introducesse numeri decimali, essi verrebbero trasformati automaticamente in interi proprio per il fatto di venir depositati in una variabile numerica intera.

LINEA 130 - Se non ci fosse questa riga, il programma continuerebbe all'infinito per il motivo detto poc'anzi. Se invece il valore introdotto in risposta alla domanda ETA'? vale zero, si passa alla linea 340. L'istruzione END in essa contenuta fa uscire dal programma. Facciamo notare che si arriva alla 340 solo tramite il rinvio che si ha in questa riga, in quanto la 330 e' quella che ricicla il tutto.

LINEA 140 - Arrivati a questo punto c'e' un test per vedere se l'eta' introdotta dall'operatore e' minore di 15, sotto cui non vogliamo scendere. Nel caso che E% risultasse minore di quel valore si otterrebbe la stampa della dicitura posta tra virgolette e si tornerebbe alla linea 120: il dato viene rifiutato e l'operatore ne e' avvertito. C'e' un'os-

servazione importante da fare, a questo proposito: l'istruzione di salto GOTO 120 deve essere messa in questa linea e non puo' trovarsi a quella successiva; infatti, in tal caso, essa verrebbe eseguita ad ogni tornata, e non solo nell'eventualita' che la condizione vista non fosse verificata.

LINEA 150 - Qui la variabile N viene incrementata di uno ad ogni giro; funge quindi da contatore. L'inizializzazione data va bene anche alla prima tornata.

LINEA 160 - Piu' avanti vengono usate due variabili per memorizzare i valori minimi e massimi delle eta' introdotte in precedenza. Ovviamente al primo giro tali valori devono coincidere col primo numero assegnato alla variabile E%. Ecco perche' in questa linea del programma MI ed MA (minimo e massimo) vengono inizializzate in tal senso.

LINEA 170 - Con questa riga si fa in modo che nella variabile MI sia contenuto sempre il valore piu' basso assegnato via via ad E%; se infatti il nuovo E% risulta inferiore ad MI, in quest'ultima variabile viene messo proprio quell'ultimo valore di E%. In caso contrario MI conserva il suo precedente valore, in quanto la condizione non e' soddisfatta ed il programma quindi prosegue oltre.

LINEA 180 - Il procedimento visto per la riga 170 viene ripetuto nei riguardi del valore massimo MA; naturalmente l'operatore di confronto (e' sostituito da).
Facciamo notare che e' lecito mescolare variabili numeriche di tipo differente; nelle linee viste finora variabili intere convivono con altre in singola precisione.

LINEA 190 - T% accumula ad ogni giro i valori di E%, sommandoli al totale precedente. Alla prima tornata T% e' uguale alla prima eta' introdotta, in quanto il valore precedente di T% e' zero (ricorda che il RUN dato per partire azzera tutte le variabili).

LINEA 200 - Ora viene calcolata la media aritmetica delle eta' introdotte; si raggiunge lo scopo dividendo T% per il numero di giri percorsi. Nel primo di essi ME% vale E%, ossia la prima eta' introdotta (anche le variabili MI, MA e T% assumono quel valore alla prima tornata).

LINEA 220 - Qui inizia la visualizzazione dei dati elaborati dal programma. Con questa istruzione si ottiene la frase tra virgolette, seguita immediatamente dal valore della variabile N.

LINEA 240 - La dicitura indicata in questa riga viene stampata alla tabulazione 5; successivamente si va alla tabulazione 18 per scrivere il valore di MI.

LINEE 250 E 260 - Lo stesso procedimento viene adottato per visualizzare i valori delle variabili MA e ME%. Mentre il valore di N varia ovviamente ad ogni giro, come pure quello di ME%, non si puo' dire lo stesso per MI ed MA: essi cambiamo solo se sono state verificate le condizioni poste alle linee 170 e 180.

LINEE DA 290 a 310 - Innanzitutto viene impartito l'ordine di scrivere in negativo, poi si stampa la dicitura PREMERE UN TASTO, quindi si torna alla scrittura in positivo; se non si mettesse REVOFF tutte le stampe successive avverrebbero in reverse.

LINEA 320 - Essa ripete lo schema gia' visto nell'esempio (7) a pagina 8 della lezione 14. Fino a che l'operatore non preme un tasto qualsiasi, il programma resta fermo a quel punto. Lo scopo e' quello di dargli il tempo di leggere i dati visualizzati.

LINEA 330 - E' quella che ricicla il tutto, rendendo il programma di tipo ricorsivo. Si tratta di un salto incondizionato alla linea 110.

Vediamo ora il programma nella versione da te digitata; il contenuto delle varie linee e' esattamente quello che avevi scritto, con i caratteri maiuscoli o minuscoli a seconda di come erano stati battuti. A lezione terminata potrai portarti la tua versione facendo l'opportuna scelta sul menu' finale. Indipendentemente dall'effettiva forma delle varie linee, esso girera' correttamente. Al massimo potrai ottenere risultati diversi nella visualizzazione dei dati (righe 220-260) se non hai rispettato la raccomandazione di non alterare gli spazi all'interno delle virgolette.

```
100'PROGRAMMA STAT1
110cls
120input"ETA'";e%
130ife%=0then340
140ife%(15thenprint"TROPPO BASSA":GOTO120
150N=N+1
160IFN=1THENMI=E%:MA=E%
170IFE%(MITHENMI=E%
180IFE%)MATHENMA=E%
190T%=T%+E%
200ME%=T%/N
210PRINT
220PRINT"NUMERO DI DATI INTRODOTTI:";N
230PRINT
240PRINT TAB(5) "ETA' MINIMA:";TAB(18);MI
250PRINT TAB(5) "ETA' MASSIMA:";TAB(18);MA
260PRINT TAB(5) "ETA' MEDIA:";TAB(18);ME%
270PRINT
280?
290REVON
300?"PREMERE UN TASTO"
310REVOFF
320X$=INKEY$:IFX$=""THEN320
330GOTO110
340END
```

La lezione e' finita. Confronta a fondo, linea per linea, il tuo programma con quello proposto; ripetiamo che, molto probabilmente, apparira' diverso, ma girera' regolarmente. Se vuoi passare direttamente alla verifica di questo, scegli la possibilita' zero offerta dal menu' finale.
Potresti poi esercitarti a cambiare il contenuto delle righe diverse da quelle proposte.

L E Z I O N E n. 16

INDICE

1 - ROUTINE E SUBROUTINE -

In informatica quando si parla di ROUTINE si intende un gruppo di istruzioni destinate a svolgere una ben determinata funzione. Ad esempio, se osserviamo il listato del programma (9) a pagina 9 della lezione 15, possiamo dire che le linee che vanno dalla 160 alla 180 formano la routine che aggiorna i valori delle variabili MI ed MA. Analogamente le righe da 220 a 260 possono essere definite come la routine che visualizza i dati; dalla 110 alla 330 si potrebbe parlare di routine per il trattamento dei dati che si riferiscono ad una persona, e cosi' via. Si tratta quindi di un concetto generale, abbastanza vago ed elastico.

Per SUBROUTINE si intende un gruppo di istruzioni che svolgono un certo compito, con la differenza che esse sono sfruttate piu' volte, partendo da diverse parti del programma.

Facciamo un esempio, ricavato dal programma appena menzionato:

(1)

```
100 'PROGRAMMA STAT1
110 CLS
120 INPUT "ETA' ";E%
130 IF E%=0 THEN 340
140 IF E%<15 THEN PRINT "TROPPO BASSA" : GOTO 120
150 N=N+1
160 IF N=1 THEN MI=E% : MA=E%
170 IF E%<MI THEN MI=E%
180 IF E%>MA THEN MA=E%
190 T%=T%+E%
200 ME%=T%/N
210 PRINT
220 PRINT "NUMERO DI DATI INTRODOTTI:";N
230 PRINT
240 PRINT TAB(5) "ETA' MINIMA:";TAB(18);MI
250 PRINT TAB(5) "ETA' MASSIMA:";TAB(18);MA
260 PRINT TAB(5) "ETA' MEDIA:";TAB(18);ME%
270 PRINT
280 PRINT
290 GOSUB 500
330 GOTO 110
340 END
500 REVON
510 PRINT "PREMERE UN TASTO"
520 REVOFF
530 X$=INKEY$ : IF X$="" THEN 530
540 RETURN
```

Come vedi, la routine che fa avanzare il programma solo se viene premuto un tasto (linee 290-320) e' stata spostata alle righe 500-530.
Al solito, questi numeri di linea sono solo esemplificativi; nulla vieta di usare i numeri a partire da 350 o da 10310.

Il contenuto della linea 290 e' cambiato e vi compare una nuova istruzione, come pure all'etichetta 540. Vediamole.

2 - GOSUB E RETURN - (VAI ALLA SUBROUTINE; RITORNA)

Alla linea 290 troviamo l'istruzione GOSUB 500; incontrandola, il computer salta alla label indicata. Si potrebbe dire che il comportamento di GOSUB e' analogo a quello di GOTO, ma con una differenza importantissima: quando il computer incontra l'istruzione RETURN, il programma riprende dall'istruzione immediatamente successiva a GOSUB. Nel nostro caso l'istruzione RETURN della linea 540 fa tornare alla riga 330.

In generale, quindi, possiamo dire che una subroutine consiste in un insieme di istruzioni che possono essere richiamate da un qualsiasi punto del programma con l'istruzione GOSUB; ogni subroutine deve terminare con la parola chiave RETURN, che causa il ritorno all'istruzione successiva a GOSUB.

Nell'esempio visto la subroutine e' posta in coda al programma principale, come si fa normalmente. In tal caso risulta evidente l'utilita' dell'istruzione END, che serve per separare le subroutine dalle istruzioni precedenti: si ha accesso alle subroutine da qualsiasi punto del programma tramite GOSUB, ma non si deve arrivare ad esse in nessun altro modo. Questa eventualita' non e' ben evidente nel programma in esame, dato che la linea 330 rinvia sempre alla 110, impedendo in tal modo al programma di proseguire; se immaginiamo di togliere la 330 e la 340, pero', il computer arriverebbe alla subroutine, la eseguirebbe fino alla linea 530 e poi andrebbe in errore alla 540, mandando il seguente messaggio:

```
(2)       RETURN WITHOUT GOSUB IN 540     (RETURN senza GOSUB in 540)
```

In un programma possono essere contenute diverse subroutine; normalmente esse sono scritte le une di seguito alle altre, non fosse altro che per motivi di ordine nel listato. Ogni subroutine risulta separata dalle seguenti proprio dall'istruzione RETURN.
Una subroutine puo' essere richiamata anche da un'altra subroutine: esse sono cioe' concatenate tra di loro. Il meccanismo di funzionamento non cambia; basta tenere presente che ogni RETURN fa tornare all'istruzione successiva alla GOSUB corrispondente.

Vediamo un esempio di quest'ultima possibilita', modificando ulteriormente la parte finale del programma STAT1:

```
(3)       ....
          290 GOSUB 500
          330 GOTO 110
          340 END
          500 REVON
          510 PRINT "PREMERE UN TASTO"
          520 REVOFF
          530 GOSUB 600
          540 RETURN
          600 X$=INKEY$ : IF X$="" THEN 600
          610 RETURN
```

La cosa non ha molto senso, in questo caso specifico, ma ci serve solo per spiegare il comportamento delle subroutine quando esse vengono concatenate.

Anziche' dilungarci in lunghe spiegazioni riportiamo l'elenco dei numeri di riga via via percorsi dal computer; seguendo sul listato potrai renderti conto di come funziona il tutto:

(4) 290 - 500 - 510 - 520 - 530 - 600 - 610 - 540 - 330

Oltre alla (2), si puo' avere anche un'altra segnalazione d'errore usando GOSUB; essa si ottiene se il numero di linea che segue GOSUB non esiste:

(5) UNDEFINED LINE # IN ... (numero di linea inesistente in ...)

Al posto dei puntini ci sara' il numero di linea contenente il rinvio errato.

3 - COME COMPATTARE UN PROGRAMMA -

Abbiamo gia' accennato altre volte che un programma puo' essere scritto in modi piu' o meno compressi. Fino ad ora abbiamo usato la forma piu' chiara possibile, che risulta essere, normalmente, anche quella maggiormente dilatata.

Questo modo di scrivere i programmi non crea problemi quando essi sono brevi, ma puo' darne allorche' le loro dimensioni sono grandi. Infatti potrebbe accadere di dover risparmiare celle di memoria per riuscire a scrivere un programma che altrimenti non potrebbe essere contenuto in macchina.

Il modo piu' semplice ed immediato per risparmiare memoria e' quello di eliminare gli spazi bianchi (escludendo ovviamente quelli che si trovano tra le virgolette); questa procedura fa risparmiare una cella di memoria per ogni spazio eliminato ed ha anche il pregio di rendere piu' veloce l'esecuzione del programma. Il prezzo da pagare e' che si ottiene un listato meno leggibile di quello di partenza.

Applichiamo questo metodo al programma STAT1 di pagina 3:

(6)
```
100 'PROGRAMMA STAT1
110 CLS
120 INPUT"ETA' ";E%
130 IFE%=0THEN340
140 IFE%<15THENPRINT"TROPPO BASSA":GOTO120
150 N=N+1
160 IFN=1THENMI=E%:MA=E%
170 IFE%<MITHENMI=E%
180 IFE%>MATHENMA=E%
190 T%=T%+E%
200 ME%=T%/N
210 PRINT
220 PRINT"NUMERO DI DATI INTRODOTTI:";N
230 PRINT
240 PRINTTAB(5)"ETA' MINIMA:";TAB(18);MI
250 PRINTTAB(5)"ETA' MASSIMA:";TAB(18);MA
260 PRINTTAB(5)"ETA' MEDIA:";ME%
270 PRINT
```

```
280 PRINT
290 GOSUB500
330 GOTO110
340 END
500 REVON
510 PRINT"PREMERE UN TASTO"
520 REVOFF
530 X$=INKEY$:IFX$=""THEN530
540 RETURN
```

Noterai che alcune linee soffrono molto, in quanto a leggibilita', per l'eliminazione degli spazi (come la 140 o la 160), mentre altre assorbono meglio la modifica (come la 240 o la 290). Si tratta comunque di fare l'abitudine alla lettura di listati compattati; quando li si usa regolarmente (come si fa di solito) la loro comprensione e' pressoche' immediata.

Quello visto non e' il solo modo di risparmiare spazio nella memoria centrale. Anche accorpare piu' linee di programma serve allo scopo; la cosa e' fattibile aumentando il numero di righe ad istruzioni multiple; pure questo procedimento fa aumentare la velocita' d'esecuzione.

Esistono poi tante altre possibilita' per compattare un programma. Ad esempio si possono spesso semplificare le istruzioni di stampa; oppure talvolta e' possibile cambiare l'ordine o il contenuto delle linee in modo da realizzare lo scopo.

A titolo esemplificativo, ecco una versione abbreviata del programma STAT1; essa e' una delle tante possibili e realizza un comportamento uguale al programma di partenza:

(7)

```
100 CLS'PROGRAMMA STAT1
120 INPUT"ETA' ";E%:IFE%=0THENEND
140 IFE%<15PRINT"TROPPO BASSA":GOTO120
150 N=N+1:IFN=1THENMI=E%:MA=E%
170 IFE%<MITHENMI=E%
180 IFE%>MATHENMA=E%
190 T%=T%+E%:ME%=T%/N
210 PRINT:PRINT"NUMERO DI DATI INTRODOTTI:"N:PRINT
240 PRINTTAB(5)"ETA' MINIMA:"TAB(18)MI:
    PRINTTAB(5)"ETA' MASSIMA:"TAB(18)MA:
    PRINTTAB(5)"ETA' MEDIA:"TAB(18)ME%:PRINT:PRINT
290 GOSUB500:GOTO110
500 REVON:PRINT"PREMERE UN TASTO":REVOFF
530 X$=INKEY$:IFX$=""THEN530ELSERETURN
```

La numerazione delle linee e' stata fatta ricollegandosi a quella precedente, in modo da poter confrontare agevolmente le due versioni.

Facciamo alcune osservazioni:

- Il contenuto delle linee 100 e 110 e' stato scambiato ed il tutto e' stato accorpato in un'unica riga. Lo scambio e' necessario perche' l'istruzione REM data all'inizio di una linea fa saltare il programma a quella successiva.
  Da notare che usando la parola chiave REM invece che la sua abbreviazione

avremmo duvuto scrivere:

```
100 CLS:REMPROGRAMMA STAT1
```

ossia avremmo dovuto mettere i due punti; in totale si sarebbero impiegati tre caratteri in piu'.

- La linea 120 e' l'unione delle 120, 130 e 340; si nota anche la soppressione della parola chiave THEN. In molti casi tale operazione e' accettata dal computer, ma e' meglio non abusarne troppo. Un caso analogo e' gia' stato esaminato nella (16) a pagina 9 della lezione 11.
- La riga 150 risulta dall'unione della 150 con la 160.
- Analogamente, la 190 e' stata ottenuta unendo la 190 e la 200; la 210 si ha unendo le 200, 220 e 230, mentre la 240 sostituisce il blocco di righe dalla 240 alla 280. In questo caso sono stati aboliti anche tutti i punti e virgola delle istruzioni di stampa. La cosa e' lecita ed e' largamente adottata; occorre pero' prestarvi molta attenzione, perche' puo' indurre a commettere errori involontari. Consideriamo infatti la seguente istruzione di stampa:

```
PRINT MA;MI;ME%
```

Se si aboliscono i punti e virgola si ottiene quanto segue:

```
PRINT MAMIME%
```

Cio' equivale a chiedere la stampa della variabile numerica intera MA%, che nel nostro programma non e' neppure definita e che quindi fornirebbe sempre il valore zero. La causa di questo comportamento e' data dal fatto che per il calcolatore l'espressione MAMIME% equivale alla sola variabile MA%, come abbiamo gia' visto alla lezione 1.

- Anche la linea 290 risulta dall'unione di due righe. E' importante notare che il ritorno dalla subroutine 500 avviene all'istruzione GOTO110 e non alla linea successiva a quella di GOSUB. Parlando del comportamento delle istruzioni GOSUB e RETURN abbiamo sempre detto che il rientro nella linea principale del programma avviene all'ISTRUZIONE (e non alla RIGA) che segue GOSUB.
- La 530 si ottiene accorpandola alla 540 per mezzo dell'istruzione ELSE. Avremmo potuto scrivere anche:

```
530 IFINKEY$=""THEN530ELSERETURN
```

ottenendo un risultato identico (come spiegato a pagina 9 della lezione 14).

Anche usando numeri di linea bassi si ottiene un risparmio di memoria; e' allora evidente che, agendo con questo metodo, il risultato migliore si ottiene rinumerando il programma facendolo partire dall'etichetta uno e procedendo con passo uno. Il comando da impartire sarebbe quindi il seguente:

```
RENUM 1,1
```

come avevamo gia' anticipato nello svolgimento dell'esercizio 3 della lezione 8, alla pagina 11.

Facciamo infine notare che digitando:

```
210PRINT
```

si ottiene poi il seguente listato:

```
210 PRINT
```

Il sistema si incarica di aggiungere automaticamente lo spazio.

ESERCIZIO n. 1

Supponiamo di avere la seguente linea di programma:

```
790 PRINT "GIACENZA:",GIACENZA%
```

Osserva questa versione:

```
790 PRINT"GIACENZA:"GI%
```

Si ottiene lo stesso risultato di stampa oppure no?

.....
.....
.....
.....

RISPOSTA

Le due stampe sono diverse a causa della soppressione della virgola dopo la frase posta tra virgolette; si sarebbero ottenuti risultati identici se al suo posto ci fosse stato un punto e virgola. La virgola causa una stampa tabulata, come abbiamo visto alla lezione 2. La sostituzione dell'identificatore GIACENZA% con GI% non comporta differenze.

---

4 - CONCATENAMENTO DI PROGRAMMI -

Puo' succedere che tutti gli artifici appena esaminati non siano sufficienti: il programma non riesce a stare nella memoria disponibile. Questo non significa che si debba rinunciare ad esso. Esiste infatti un'altra scappatoia, che e' stata concepita per risolvere questo tipo di problemi: bisogna frazionare il programma in due o piu' parti, collegandole poi tra di loro con l'istruzione RUN.

In pratica si procede come segue: si digita la parte di programma che puo' essere contenuta in memoria e la si registra su disco assegnandole un nome (ad esempio PARTE1). Si scrive poi il resto del programma e lo si salva su disco col suo nome (PARTE2, per continuare nel nostro esempio).
E' evidente che lanciando il primo programma col comando RUN"PARTE1" esso va in esecuzione, ma al suo termine non si ha l'aggancio con PARTE2; cio' si ottiene

se l'ultima istruzione del primo programma e':

```
(8)       2730 RUN "PARTE2"
```

(il numero di riga e' scelto a caso, come sempre). Arrivato a quel punto, il computer va a cercare sul floppy il programma di nome PARTE2, lo carica in memoria e lo manda automaticamente in esecuzione.

Tale processo puo' essere ripetuto tante volte quante si vuole; si puo' anche riciclare il primo programma, mettendo come ultima istruzione di quello finale RUN "PARTE1".

Tutto questo, pero', ha un limite: l'istruzione RUN azzera il valore di tutte le variabili; quindi non e' possibile, con questo metodo, travasare direttamente i loro valori da un programma a quello che gli e' concatenato.
Per trasmettere le variabili occorre quindi ridefinirle all'inizio della parte concatenata. Anche questo procedimento presenta delle limitazioni: vale solo nel caso che esse assumano valori costanti noti a priori. In caso contrario non potremmo definire dei valori che non conosciamo.
Questo e' il caso piu' sfortunato, ed ha comunque una soluzione: bisogna registrare su disco, in un file apposito, i valori delle variabili da trasportare; successivamente si aggancia il programma seguente, che conterra' al suo inizio le istruzioni per leggere il file in oggetto. Tale procedimento e' piu' laborioso da dire che da fare; sara' comunque piu' chiaro dopo che avremo esaminato il trattamento dei file.

Qui finisce la lezione. Dopo la stampa del riepilogo fa il seguente esercizio: carica in memoria il programma AREA1 che hai scritto alla lezione 10; esegui poi su di esso una modifica, introducendo la seguente linea:

```
25 IF C1=0 THEN RUN "STAT1"
```

In questo modo potrai eseguire il programma AREA1 fino a che non assegnerai il valore zero al primo cateto; quando questo accadra', la riga 25 diventa operante e causa l'aggancio col programma STAT1. Salva il nuovo AREA1 su un disco vergine, in modo da poter fare liberamente tutte le prove necessarie. Sullo stesso disco registra poi anche il programma STAT1 (nella versione che hai digitato alla lezione 15), dopo averlo modificato in modo che arrivati alla linea 340 esso agganci nuovamente il programma AREA1.

RIEPILOGO DELLA LEZIONE 16

GOSUB

L'istruzione GOSUB seguita da un numero di linea fa eseguire la subroutine che inizia a quella riga. Ogni subroutine puo' essere usata tutte le volte che risulta necessario, partendo da punti qualsiasi del programma.

RETURN

L'istruzione RETURN pone fine alla subroutine e fa tornare il controllo del programma all'istruzione immediatamente seguente alla GOSUB di partenza.

RISPARMIO DI MEMORIA

Si puo' risparmiare memoria eliminando tutti gli spazi superflui; semplificando, nei limiti del possibile, le operazioni di stampa; raggruppando piu' linee in una sola; togliendo il carattere delle virgolette quando si trova a fine riga; modificando opportunamente il contenuto delle varie linee del programma. Questi metodi generalmente raggiungono anche lo scopo di rendere piu' veloce l'elaborazione dei dati.

CONCATENAMENTO DI PROGRAMMI

Un programma puo' aggianciarne un altro per mezzo dell'istruzione RUN; si perdono pero' i valori delle variabili.

LEZIONE n.17

INDICE

1 - LPRINT - (SCRIVI SU STAMPANTE)

Le istruzioni LIST e LLIST sono perfettamente analoghe; l'unica differenza consiste nel fatto che la prima ha effetto sul monitor mentre la seconda agisce sulla stampante. Una situazione simile si ha con PRINT e LPRINT.

L'istruzione LPRINT serve per scrivere dati con la stampante; per essa valgono tutte le considerazioni svolte per PRINT. Ecco qualche esempio:

```
(1)     LPRINT "Nuova Elettronica"
        LPRINT H3;K$,AA$
        LPRINT
        LPRINTCHR$(67)
        LPRINT TAB(8)"MESE DI ";M$
        LPRINT 1" - VENDITE A RATE: "VR%
```

Come vedi, valgono tutte le regole viste per PRINT, compresa la soppressione del punto e virgola per stampe di dati consecutivi.
Il terzo caso serve per saltare una riga di stampa; a questo proposito c'e' da osservare che non tutte le stampanti si comportano allo stesso modo. Ad esempio, con stampanti EPSON non ci sono problemi, mentre con modelli OKI (MICROLINE 80) l'istruzione LPRINT data da sola viene ignorata. Per riuscire a lasciare righe si stampa vuote occorre impartire la seguente istruzione:

```
(2)     LPRINT" "
```

ossia bisogna fare stampare uno spazio. Chi usa questo tipo di stampante puo' definire all'inizio di un programma una stringa contenente lo spazio, e poi richiedere la stampa di quella stringa tutte le volte che serve lasciare una riga in bianco:

```
(3)     ....
        30 R$=" "
        ....
        260 LPRINTR$
        ....
```

Anche le istruzioni contenenti TAB possono dare qualche differenza tra una stampante e un'altra; basta comunque fare qualche semplice prova per rendersi conto di come vanno le cose con quella che si ha a disposizione.

La quasi totalita' delle stampanti dispone di un tasto o di un interruttore per metterle in linea col computer, ossia per abilitarle o disabilitarle alla stampa. Se si impartisce un LPRINT con la stampante che non e' in linea, il computer non puo' proseguire. Ci sono solo due modi per uscire da questa situazione: o si attiva la stampante, o si fa un RESET; infatti in un caso simile il BREAK non e' operante. Normalmente, invece, se la stampante e' collegata col cavo ma e' spenta, il computer riesce ad eseguire istruzioni di LPRINT (naturalmente solo al suo interno) e quindi il programma procede come se la stampa fosse realmente avvenuta.

Si ricade invece nell'inghippo precedente anche se il cavo che collega la stam-

pante al computer risulta staccato o difettoso, o se la stampante .... non c'e' proprio!

2 - STR$ - (da STRING=STRINGA)

In questa lezione esamineremo tutte le rimanenti funzioni di stringa, ad esclusione di quelle che si riferiscono al trattamento dei file di tipo random. Cominciamo con l'esaminare l'istruzione STR$; con essa si trasforma un numero in una stringa, ossia si trasferiscono segno e cifre all'interno di una variabile di stringa: si puo' anche dire che con STR$ un tipo numerico diventa di tipo alfanumerico.

Vediamo un esempio:

```
(4)       A=206.38 : A$=STR$(A)
```

Con la prima istruzione si deposita nella variabile numerica A il numero 206.38; con la seconda si mette nella variabile di stringa A$ il numero 206.38, compreso il segno. Per chiarire bene le idee, con la seguente istruzione di stampa:

```
          PRINT "*";A;"*";A$;"*"
```

otterremmo quanto segue:

```
          * 206.38 * 206.38*
```

La stampa della variabile A avviene, come gia' sai, lasciando uno spazio iniziale riservato al segno, ed uno in coda al numero. Nella variabile di stringa A$ va invece a finire tutto A tranne che lo spazio finale. Si puo' verificare quanto detto anche in altro modo:

```
          PRINT LEN(A$)
```

Con questa istruzione si ottiene il numero 7 (somma delle cinque cifre piu' lo spazio del segno ed il punto che separa la parte intera da quella decimale).

Ecco qualche altro esempio (tra parentesi il contenuto della variabile alfanumerica ottenuta):

```
(5)       C3%=-7 : E$=STR$(C3%)                              (E$="-7")
          T(5)=3.1 : D=2 : LL$=STR$(T(5)*D)                  (LL$=" 6.2")
          E#=-1.0000008 : E!=-2 : I$(4)=STR$(E#*E!)          (I$(4)=" 2.000
          S$=STR$(4*-6/-2)                                   (S$=" 12")
          A$=STR$(4*-6/-2) : A$=RIGHT$(A$,LEN(A$)-1)         (A$="12")
```

L'istruzione STR$ richiede dunque la specifica di un parametro tra parentesi; esso deve essere di tipo numerico e puo' essere dato anche sotto forma di espressione algebrica.

Gli ultimi due esempi mostrano come si possa ottenere all'interno della stringa il solo numero senza lo spazio riservato al segno: dopo aver trasformato il nu-

mero in stringa, di questa si prendono tutti i caratteri di destra fino al segno escluso (ossia un numero di caratteri uguali alla lunghezza della stringa ottenuta meno uno).

ESERCIZIO n. 1

Quale valore assume la variabile alfanumerica J$ definita come segue?

```
J$=MID$(STR$(7*-4),2,1)
```

. . . . . .
. . . . . .
. . . . . .
. . . . . .

RISPOSTA

L'istruzione STR$(7*-4) fornisce la stringa "-28"; prelevando da essa un carattere partendo dal secondo, si ha come risultato "2".

---

3 - VAL - (da VALUE=VALORE)

L'istruzione VAL esplica una funzione esattamente inversa a quella di STR$: essa trasforma una stringa in in numero. Vediamo un esempio:

```
(6)     D$="11.8902" : B=VAL(D$)
```

Nella variabile numerica B va a finire il numero 11.8902, che in partenza era il contenuto di una stringa.
Riportiamo qualche altro esempio:

```
(7)     AA$="-56" : M=VAL(AA$)                          (M=-56)
        E=VAL("3+5")                                    (E=3)
        Z$="3" : X$="5" : D=VAL(Z$+X$)                  (D=35)
        C$="diodo" : B$="22" : E=VAL(C$+B$)             (E=0)
        M$="-1" : ND$="+4" : A=VAL(M$+ND$)              (A=-1)
```

Esaminandoli attentamente si puo' ricavare la seguente regola:
- l'istruzione VAL parte dall'inizio della stringa specificata e deposita nella variabile numerica indicata tutte le cifre che incontra, eventualmente precedute dal segno; il primo carattere non numerico pone termine alla formazione del numero. Pertanto se il primo carattere della stringa non e' una cifra, nella variabile numerica viene depositato il valore zero.

Nel secondo caso, ad esempio, la cifra 3 entra nella variabile E, poi il processo si ferma perche' segue un carattere non numerico (il segno +).

Per tutti gli altri casi valgono considerazioni analoghe, fatte nel rispetto della regola anzidetta.

ESERCIZIO n. 2

Osserva la seguente istruzione:

```
F$="-13,5 Gradi Centigradi" : GC!=VAL(F$)
```

Nella variabile GC! quale valore viene depositato?

.....
.....
.....
.....

RISPOSTA

A parte il segno iniziale, il primo carattere non numerico incontrato e' la virgola; pertanto in GC! va a finire il numero precedente, ossia -13. Si sarebbe ottenuto -13.5 se al posto della virgola ci fosse stato il punto.

---

4 - STRING$ -

L'istruzione STRING$ necessita di due parametri posti tra parentesi e serve per generare delle stringhe formate da caratteri tutti uguali tra di loro.
Vediamo subito un esempio:

```
(9)     V$=STRING$(38,"*")
```

Con questa istruzione si deposita nella variabile V$ una stringa formata da 38 asterischi.

Quando si usa STRING$ bisogna quindi specificare un parametro numerico (che indica quanti caratteri si intende considerare) ed un parametro alfanumerico (che specifica il carattere che si intende usare per formare la stringa).

Ecco qualche altro esempio:

```
(10)    B3$=CHR$(35) : C$=STRING$(80,B3$)
        A$=STRING$(25,CHR$(134))
        Y$=STRING$(32,"OCHE")
```

Esaminiamoli in dettaglio. Dato che il carattere corrispondente al codice ASCII 35 e' il simbolo #, col primo caso si ottiene in C$ una stringa formata da 80 caratteri tutti uguali a quello.
Il secondo esempio deposita in A$ una stringa composta dai 25 caratteri grafici

corrispondenti al codice 134.
L'ultimo caso non fornisce molte oche, ma una stringa di 32 caratteri uguali alla prima lettera di quella parola (O maiuscola).
Sul monitor, comunque, puoi vedere i valori delle tre variabili formate da quelle istruzioni.

Dall'ultimo esempio si ricava che anche assegnando al parametro alfanumerico una lunghezza di diversi caratteri, il sistema prende in considerazione solo il primo di essi.

ESERCIZIO n. 3

Qual'e' il modo piu' breve per depositare nella variabile alfanumerica P$ una stringa formata da 40 puntini?

.....
.....
.....
.....

RISPOSTA

Per assegnare a P$ una stringa formata nel modo richiesto, si deve scrivere:

```
P$=STRING$(40,".")
```

Al posto delle virgolette contenenti il punto si potrebbe impiegare anche il codice ASCII corrispondente, ossia CHR$(46); questo pero' non e' il metodo piu' breve.

ESERCIZIO n. 4

Supponiamo di avere la seguente stringa:

```
K$="CHIAVE"
```

Vogliamo utilizzarla per formare un'altra stringa T$ composta da 15 lettere H maiuscole; come si deve definire T$?

.....
.....
.....
.....

RISPOSTA

Il modo migliore per isolare la lettera H dalla parola CHIAVE e' quello di usare

la funzione MID$; allora T$ puo' essere definito in questo modo:

```
T$=STRING$(15,MID$(K$,2,1))
```

E' chiaro che l'istruzione:

```
T$=STRING$(15,"H")
```

avrebbe raggiunto lo scopo in modo piu' veloce; l'esercizio proposto ha lo scopo di renderti familiare l'uso di diverse funzioni e di costringerti ad una certa elasticita' mentale.

---

5 - INSTR - (da IN STRING=DENTRO LA STRINGA)

INSTR e' l'ultima funzione di stringa che ci rimane da considerare; essa fornisce un valore numerico che indica in quale posizione si trova una stringa all'interno di un'altra. Spieghiamoci con un esempio:

```
(11)    A$="PERSONAL COMPUTER" : S=INSTR(A$,"AL")
```

Nella variabile numerica S viene messo il valore 7, in quanto AL e' contenuto in A$ e si trova al settimo carattere partendo da sinistra.

Manteniamo per A$ il precedente valore; con l'istruzione:

```
S=INSTR(A$,"al")
```

otteniamo per S il numero zero, in quanto le lettere specificate non sono contenute in A$. Anche i gruppi 'Al' e 'TER ' avrebbero dato lo stesso risultato, in quanto non esistono in quella forma.

Qui di seguito riportiamo qualche altro esempio; tra le parentesi i numeri forniti dalla funzione INSTR:

```
(12)    P=INSTR("BOLOGNA","LO")                          (P=3)
        E$=RIGHT$("INTEGRATO",5) : H=INSTR(E$,"TE")      (H=0)
        U$=STRING$(6,CHR$(100)) : W=INSTR(U$,"d")        (W=1)
```

Il primo caso non necessita di spiegazioni; il secondo fornisce il valore zero perche' le lettere TE non sono contenute in E$ (dal momento che in E$ e' depositata la parola GRATO).
Nell'ultimo esempio, dato che al codice ASCII 100 corrisponde la lettera D minuscola, si ottiene in W il numero 1 (infatti si ha: U$="dddddd").

Nell'uso delle funzioni di stringa esaminate, puo' accadere di commettere diversi tipi di errore, come sbagliare l'uso delle parentesi, oppure usare parametri in forme non appropriate; si possono quindi ottenere diverse segnalazioni d'errore.

Le piu' frequenti sono queste:

```
(13)      SYNTAX ERROR
          ILLEGAL FUNCTION CALL
          TYPE MISMATCH
```

che abbiamo gia' incontrato in altre occasioni. Non ci soffermiamo quindi piu' del necessario su di esse.

ESERCIZIO n. 5

Abbiamo due stringhe definite nel modo seguente:

```
A$="PROGRAMMARE " : B$=" IN BASIC"
```

Che numero si ottiene col comando: PRINT INSTR(A$+B$,"SIC") ?

.....
.....
.....
.....

RISPOSTA

Il gruppo di lettere richieste si trova nella stringa somma delle due assegnate alla posizione 19, cominciando da sinistra (bisogna conteggiare anche tutti gli spazi).

Facendo ancora riferimento alle due stringhe precedenti, che numeri si ottengono coi due comandi seguenti?

```
PRINT INSTR(A$+B$,"mare")
PRINT INSTR(A$+B$,"MARE")
```

.....
.....
.....
.....

RISPOSTA

Il gruppo di lettere MARE scritto in caratteri minuscoli non esiste all'interno delle stringhe assegnate, quindi il primo comando restituisce il numero zero. Le stesse lettere, invece, esistono in caratteri maiuscoli, e si ottiene il valore 8.

Probabilmente non riuscirai, per il momento, a renderti conto dell'effettiva utilita' di tutte le funzioni adibite al trattamento delle stringhe.

Avrai comunque notato che sono numerose e molto potenti; mano a mano che proce-derai nella strada della programmazione in BASIC, ne farai un uso sempre piu' intenso ed appropriato. Anche nel prosieguo di questo corso d'istruzione vedremo altre applicazioni interessanti.

Come abbiamo gia' accennato, restano da considerare solo le funzioni di stringa che sono destinate al trattamento dei file random; esse sono sei in tutto:

CVD
CVI
CVS
MKI$
MKD$
MKS$

e le incontreremo piu' avanti.

La lezione e' finita. A riepilogo terminato, dovresti esercitarti a lungo sia nell'uso dell'istruzione LPRINT che delle varie funzioni per il trattamento del-le stringhe. Puoi cominciare con l'eseguire gli esempi proposti nel corso della presente lezione, per passare poi ad altri di tua invenzione.

RIEPILOGO DELLA LEZIONE 17

LPRINT

L'istruzione LPRINT serve per scrivere con la stampante. Il suo comportamento e' del tutto simile a quello di PRINT.

STR$

La funzione di stringa STR$ trasforma il contenuto della parentesi (che deve essere un numero) in una stringa; tale stringa ha per primo carattere il segno del numero, mentre non contiene il carattere vuoto terminale che e' associato ad esso.

VAL

La funzione VAL esplica un lavoro inverso a quello di STR$: trasforma una stringa in un numero. In questo processo sono coinvolti solo i caratteri a sinistra della stringa; esso si arresta al primo carattere che non sia una cifra.

STRING$

STRING$ serve per creare stringhe formate da caratteri tutti uguali. Necessita di due parametri; il secondo e' alfanumerico e specifica il carattere che si intende usare; il primo e' numerico e dice quanti di quei caratteri si vogliono prendere. Nell'uso occorre fare attenzione al numero di caratteri riservati alle stringhe con l'istruzione CLEAR.

INSTR

La funzione INSTR restituisce un valore numerico; esso indica la posizione a cui si trova una stringa all'interno di un'altra. I due parametri da specificare sono quindi entrambi di tipo alfanumerico.

L E Z I O N E n. 18

INDICE

1 - COS'E' UN FILE -

In questa lezione cominceremo a parlare dei file sequenziali, ossia affronteremo uno degli argomenti piu' importanti: l'archiviazione dei dati.

Per coloro che si trovano alle prime esperienze di programmazione il concetto di file, probabilmente, non e' del tutto chiaro. Innanzitutto possiamo dire che un programma registrato su disco e' un file; hai gia' fatto qualche esperimento in proposito. In particolare hai senza dubbio notato che ogni file-programma deve essere contraddistinto da un nome; e' per mezzo di esso che quel programma viene individuato e trattato nel modo voluto.

I programmi, pero', comportano quasi sempre l'elaborazione e l'archiviazione dei dati che prendono in esame. Pensiamo ad esempio ad un programma che gestisce gli articoli in giacenza in un magazzino di parti di ricambio per automobili; esistera' un archivio in cui sono registrati i dati caratteristici di ogni pezzo (come il numero di codice e la descrizione) assieme alla quantita' presente in magazzino. Ogni operazione di vendita o di acquisto di quel ricambio comporta un aggiornamento del numero che esprime la quantita' disponibile. L'addetto al computer, quando batte alla tastiera le vendite o gli acquisti via via effettuati, va ad aggiornare l'archivio delle giacenze. Con una procedura simile e' quindi possibile tenere sotto controllo la situazione di tutti gli articoli trattati.
E' evidente che il programma che gestisce il magazzino e' concepito in questo modo: assume i dati digitati dall'operatore, li elabora nel modo opportuno e va a registrare in uno o piu' file-archivio i dati aggiornati.

Quella descritta e' la situazione tipica che si presenta nel trattamento dei dati con un elaboratore elettronico, e fa perno sulla gestione degli archivi; si capisce quindi facilmente come l'argomento che stiamo trattando sia uno dei pilastri su cui poggia tutta la costruzione della programmazione.

I file possono quindi essere di due tipi: file-programma e file-archivio; questi ultimi, a loro volta, possono essere suddivisi in due categorie, a seconda della loro struttura. Si parla quindi di file sequenziali e file random; vediamoli in dettaglio.

FILE SEQUENZIALI - I due tipi di archivio esistenti differiscono per il modo con cui si puo' arrivare al dato desiderato. I file ad accesso sequenziale sono organizzati nel modo seguente: tutti i dati sono registrati uno dopo l'altro, e si puo' giungere ad uno intermedio solo leggendo l'archivio dall'inizio, in sequenza. Se ci interessa conoscere, ad esempio, il contenuto del tredicesimo dato, dovremo iniziare la consultazione dell'archivio leggendo il primo, poi passare al secondo, e cosi' via fino a quello che ci serve.
Anche l'aggiornamento dei valori in archivio deve seguire la stessa regola, e puo' avvenire solo registrando ex novo tutti i dati, dal primo all'ultimo.

FILE RANDOM - La parola random, in Inglese, significa casuale; si puo' quindi intuire quale sia la differenza rispetto ai file sequenziali: in questo tipo di archivio ogni dato e' accessibile in modo diretto, sia in lettura che in scrittura. Cio', tra la'altro, significa che un aggiornamento dei valori puo' essere fatto andando a registrare solo quelli nuovi, lasciando intatti tutti gli altri.

In assoluto non si puo' affermare che un tipo di file sia preferibile ad un altro; la scelta dipende dal problema in oggetto, e va effettuata caso per caso.

2 - COS'E' UN RECORD -

Torniamo all'esempio precedente di un'attivita' commerciale riguardante pezzi di ricambio per auto; supponiamo di voler tenere in archivio, per ogni articolo, i seguenti dati: numero di codice, descrizione, numero dei pezzi dispinibili a magazzino, prezzo unitario IVA esclusa, aliquota IVA, prezzo unitario comprensivo di IVA, tipo di veicolo cui si riferisce, note varie. Ogni articolo e' quindi contraddistinto da questi otto parametri, il cui insieme forma un RECORD, ossia una raccolta di dati (di natura anche diversa) che formano una struttura unica.

Il catalogo dei ricambi potrebbe avere questo aspetto:

CODICE	DESCRIZIONE	N.	LIRE	+IVA	MODELLO	NOTE
00283551	ALBERO TRASMISSIONE	1	57600	18 67968	ALFA ROMEO 2000 GTV	modello 1976
00772438	FANALE POST. SIN.	16	32000	18 37760	FIAT 500	ultimo tipo
20075922	FUSIBILI 16 AMPERE	258	2300	18 2714	-	scatola 100 pezzi
20075923	FUSIBILI 25 AMPERE	105	2750	18 3245	-	scatola 100 pezzi

Ognuna delle righe dell'esempio forma il record di dati di un articolo; essa contiene sia valori numerici che alfanumerici, opportunamente trattati dal programma che li gestisce.

Per concludere questa introduzione, ricordiamo che FILE, in Inglese, non significa solo fila; sta infatti ad indicare anche un insieme di cartelle ordinate in appositi raccoglitori da ufficio (per argomenti, in ordine alfabetico, ecc.), ossia un insieme di informazioni strutturate in qualche modo. Questo secondo significato ci sembra esprimere adeguatamente il concetto di file-archivio.

2 - TRATTAMENTO DEI FILE SEQUENZIALI (prima parte) -

Abbiamo gia' detto che i file sono raccolte strutturate di informazioni; essi vengono registrati su dischetto. Per poter accedere ad un file, indipendentemente dal tipo o dal fatto di effettuare operazioni di lettura o scrittura, occorre mettere in comunicazione il computer con i floppy. Tale canale serve per farvi transitare i dati e si chiama BUFFER. In pratica si tratta di una porzione di memoria centrale che svolge funzioni di collegamento e di scambio (infatti buffer significa zona cuscinetto, respingente con molla; anche tale dicitura esprime adeguatamente le funzioni svolte, come in genere accade per tutta la terminologia informatica).

La prima regola da rispettare per usare qualsiasi tipo di file e' quindi quella di creare il buffer di comunicazione; l'istruzione di apertura di un file svolge

proprio questa funzione. C'e' da dire anche che i file aperti possono essere in numero superiore ad uno: si possono cioe' gestire diversi archivi contemporaneamente, a patto, ovviamente, di riservare ad ognuno di essi il proprio buffer.
Il Basic e' in grado di trattare fino ad un massimo di 15 buffer; il sistema riserva automaticamente tre buffer per la gestione dei file, e questo numero e' generalmente sufficiente. E' comunque possibile, per mezzo del comando BASIC del DOS, fare delle variazioni al riguardo: si puo' decidere di usarne un numero maggiore o di non averne bisogno affatto. Vedremo in altra occasione come si usa il comando in questione; ci preme solo far notare ancora che ogni buffer sottrae una porzione di memoria, quindi e' consigliabile non aprirne troppi in contemporanea, se cio' non e' imposto da particolari esigenze: meglio riservare al programma un maggior numero di locazioni di memoria, magari in previsione di futuri ampliamenti, piuttosto che sprecarne inutilmente per l'uso di file che poi non saranno utilizzati.

Si puo' aprire un file in quattro modi diversi (tre servono per i file sequenziali ed uno per quelli random). Per fissare le idee, vediamo come si deve operare per poter registrare dei dati in un archivio sequenziale:

```
(1)       OPEN"O",1,"ESEMPIO"
```

Bisogna quindi mettere la parola chiave OPEN, seguita dalla lettera O tra virgolette; fino a questo punto abbiamo ordinato al computer di aprire (OPEN) un file sequenziale in scrittura (O sta per OUTPUT, ossia uscita: i dati escono dalla memoria centrale per essere travasati sul disco).
Seguono poi una virgola, un numero, un'altra virgola ed un nome tra virgolette. il numero indica quale dei tre buffer che si hanno normalmente a disposizione si intende usare; il nome e' obbligatorio, e puo' essere formato da un massimo di otto caratteri. Il primo di essi deve essere una lettera; i caratteri eventualmente presenti oltre l'ottava posizione vengono ignorati dal computer (non si ha quindi una segnalazione d' errore, ma un troncamento della parte finale).
Dopo aver aperto il file, si possono scrivere in esso dei dati con un'istruzione del tipo seguente:

```
(2)       PRINT#1,G;S%
```

Come si vede, dopo la parola chiave PRINT (abbreviabile col punto interrogativo in sede di digitazione) bisogna aggiungere il carattere # seguito dal numero di buffer attivato con l'apertura; seguono poi i dati da registrare, che possono essere sia di tipo numerico che alfanumerico (nell'esempio si registra prima il numero depositato nella varabile numerica in semplice precisione G, poi il contenuto della variabile numerica intera S%).

La registrazione dei dati sul disco con l'istruzione PRINT# avviene seguendo le stesse regole della scrittura sul monitor con l'istruzione PRINT. Tale asserzione e' valida solo per la registrazione di valori numerici, in quanto il trattamento delle stringhe e' piu' complesso, come vedremo tra poco.
Nel caso appena visto tra le due variabili numeriche e' stato posto un punto e virgola: cio' causa una registrazione del numero corrispondente a S% in coda al valore di G (con le solite considerazioni sul carattere iniziale riservato al segno e quello finale bianco). Se mettiamo una virgola al posto del punto e virgola il valore di S% non viene scritto attaccato al precedente, ma tabulato la-

sciando un certo numero di caratteri in bianco, esattamente come per le istruzioni di PRINT sul monitor (vedere i vari esempi fatti in proposito). I dati vengono ugualmente registrati nell'archivio, ma si spreca una notevole quantita' di posto sul dischetto; in pratica, quindi, la registrazione di dati numerici viene fatta interponendo sempre tra un valore e l'altro un punto e virgola.

Affrontiamo ora il trattamento delle stringhe in registrazione, che risulta alquanto piu' complesso che non quello dei numeri.
Senza dilungarci in noiose spiegazioni teoriche, per ora ci sembra sufficiente quanto segue: mentre i valori numerici sono registrati nel modo appena visto, quelli alfanumerici vanno separati gli uni dagli altri da caratteri delimitatori, che possono essere di due tipi: virgole o virgolette. Vediamo innanzitutto il primo caso, col seguente esempio:

```
(3)       PRINT#1,FE$;",";H$
```

In questo modo sul disco viene registrato il contenuto di FE$, poi il carattere della virgola e quindi la stringa corrispondente alla variabile H$; i punti e virgola si incaricano di non fare sprecare spazio.
L'esempio fatto vale pero' solo se le stringhe trattate non contengono virgole; infatti con quel sistema di registrazione i dati alfanumerici risultano separati gli uni dagli altri proprio dal carattere della virgola. Quindi se esso fosse contenuto anche all'interno delle stringhe in oggetto, tutto il sistema verrebbe scombussolato; vediamo il perche' con tre esempi:

```
(4)       OPEN"O",1,"SEQ1"
          PRINT#1,"ROSSI MARIO";"VIA LARGA 173 - MI"
          CLOSE#1
                    (ROSSI MARIOVIA LARGA 173 - MI)

(5)       OPEN"O",1,"SEQ2"
          PRINT#1,"ROSSI MARIO";",";"VIA LARGA 173 - MI"
          CLOSE#1
                    (ROSSI MARIO,VIA LARGA 173 - MI)

(6)       OPEN"O",3,"SEQ3"
          PRINT#3,"ROSSI MARIO";",";"VIA LARGA, 173 - MI"
          CLOSE
                    (ROSSI MARIO,VIA LARGA, 173 - MI)
```

Prima di procedere spieghiamo il significato dell'istruzione CLOSE: essa serve per chiudere il buffer attivato con la precedente OPEN, in modo che esso torna ad essere disponibile per altri file; quando le operazioni di scrittura o lettura di un archivio sono terminate, occorre effettuare la chiusura del buffer ad esso corrispondente.
Possiamo usare CLOSE in due modi, evidenziati dagli esempi: se CLOSE e' seguita dal carattere # e da un numero, si ordina al computer di chiudere solo il file ad esso corrispondente; altri file, eventualmente aperti, restano in tale condizione. Una chiusura fatta invece come nell'ultimo caso ha effetto su tutti i file che risultano aperti a quel momento.

Per la registrazione dei dati alfanumerici abbiamo usato una forma diretta, in-

vece che passare attraverso variabili di stringa; in ognuno dei tre esempi scriviamo su disco le due frasi poste tra le virgolette. Sotto ad ogni esempio e' riportato cio' che viene registrato fisicamente sul supporto magnetico.

Vediamo ora come si effettua un'operazione di lettura dei dati in uno qualsiasi degli archivi appena registrati; cominciamo dal secondo:

```
(7)     OPEN"I",1,"SEQ2"
        INPUT#1,L1$,L2$
        CLOSE
        PRINT L1$ : PRINT L2$
```

La prima riga e' simile alla (1), tranne che la lettera usata per qualificare il tipo di apertura del file SEQ2 e' la I; cio' indica al computer che si tratta di un'operazione di prelievo di dati dal dischetto (I sta per INPUT, ossia ingresso di valori nella memoria centrale). E' intuitivo che dopo aver fatto una simile apertura del file, potremo solo effettuare operazioni di lettura dei dati; l'istruzione adibita a questo scopo e' la INPUT, che funziona in modo analogo alla PRINT gia' usata in scrittura. La seconda riga dell'esempio ordina al computer di leggere due dati di stringa, prelevandoli dal buffer numero 1, ossia dal file SEQ2.
Dopo aver chiuso il file, si chiede la stampa delle due variabili alfanumeriche usate in lettura; sul monitor si legge quanto segue:

```
(8)     ROSSI MARIO             (L1$)
        VIA LARGA 173 - MI      (L2$)
```

Questo risultato si ha perche' il carattere della virgola (sul disco) e' un separatore di stringhe. In pratica quindi un'operazione di lettura di stringhe sequenziali funziona in questo modo: INPUT#1,L1$ fa partire la ricerca dall'inizio del file SEQ2 e deposita in L1$ tutti i caratteri che via via si incontrano prima di arrivare ad una virgola. Essa funge da separatore e non compare ne' nella stringa precedente ne' in quella successiva. La richiesta del secondo dato (L2$) fa entrare caratteri a cominciare dal primo dopo la virgola, fino ad arrivare ad un'altra virgola o alla fine del file.
Noi siamo in quest'ultima evenienza, ed otteniamo per L1$ e L2$ le frasi scritte poco sopra.

Se immaginiamo di ripetere l'insieme delle istruzioni (7) al file SEQ3 creato con la (5), otteniamo queste due stringhe:

```
(9)     ROSSI MARIO
        VIA LARGA
```

La spiegazione della perdita della parte finale dell'indirizzo sta nella virgola che separa il nome della via dal numero civico; quel carattere diventa separatore di stringa e tronca la formazione dell'indirizzo.

Le cose vanno ancora peggio applicando la (7) al file SEQ1 scritto con la (4): si ottiene infatti la segnalazione d'errore:

```
(10)    INPUT PAST END      (INPUT oltre la fine del file)
```

Cosa e' successo? Dopo quanto abbiamo detto finora, la spiegazione dovrebbe essere abbastanza chiara: il computer parte dall'inizio del file SEQ1 e deposita nella prima variabile di stringa L1$ tutti i caratteri che precedono la prima virgola, dal momento che quello e' il carattere separatore. Il fatto e' che sul disco e' stato registrato cio' che compare tra parentesi nella (4): non ci sono virgole, quindi quel modo di registrare i dati ha in effetti creato un'unica stringa:

```
(11)      ROSSI MARIOVIA LARGA 173 - MI
```

L'intera dicitura va a finire in L1$. Il computer poi prosegue, cercando di leggere altri caratteri per formare L2$; essi pero' sono terminati, in quanto gia' con L1$ abbiamo letto l'intero contenuto del file. La (10) ci avvisa dell'errore commesso in lettura (in effetti poi esso era avvenuto in scrittura, dal momento che il metodo usato ha portato alla fusione delle due stringhe).

Da quello che si e' detto finora si conclude che per poter trattare le stringhe occorre che esse non contengano virgole al loro interno, in quanto falserebbero le operazioni di lettura; inoltre le varie stringhe vanno separate le une dalle altre usando il carattere della virgola. Cio' puo' essere fatto sia in forma diretta (come negli esempi (5) e (6)), che in altri due modi. Scriviamo l'esempio (5) in queste due nuove versioni:

```
(12)      OPEN"O",1,"SEQ2"
          PRINT#1,"ROSSI MARIO";CHR$(44);"VIA LARGA 173 - MI"
          CLOSE#1

(13)      V$=","
          OPEN"O",1,"SEQ2"
          PRINT#1,"ROSSI MARIO";V$;"VIA LARGA 173 - MI"
          CLOSE#1
```

Nel primo caso alla virgola si e' sostituito il codice ASCII corrispondente, facendo uso della funzione di stringa CHR$. Nell'altro esempio, invece, si definisce all'inizio la stringa V$ depositando in essa la virgola; poi quando bisogna registrare la virgola si usa V$; e' evidente che quest'ultimo procedimento e' il piu' vantaggioso quando si devono registrare molte stringhe, in quanto si possono risparmiare parecchie battute alla tastiera.

Nella prossima lezione vedremo che c'e' anche il modo di registrare in file sequenziali stringhe contenenti la virgola; come carattere separatore si puo' infatti usare anche quello delle virgolette.

La lezione 18 e' finita. Potresti esercitarti nell'uso delle istruzioni di gestione dei file viste finora. Questi concetti possono sembrare un po' ostici all'inizio, ma diventano poi d'uso immediato dopo qualche esercizio pratico. Con la prossima lezione termineremo la parte teorica del trattamento dei file sequenziali e nella vetesima scriverai un programma pratico che vertera' su questo argomento.

RIEPILOGO DELLA LEZIONE 18

FILE

In generale un file puo' essere definito come un insieme di dati organizzati registrato su disco.
I file possono essere suddivisi in due categorie: programmi ed archivi. Questi ultimi, a loro volta, possono essere ad accesso sequenziale o casuale (random).

RECORD

Ogni file-archivio e' costituito da un gruppo di informazioni collegate tra di loro e formanti un'unica entita' logica: il record. Esso puo' contenere uno o piu' dati di tipo diverso.

BUFFER

In senso lato si chiama buffer una porzione di memoria riservata allo scambio di dati tra diverse parti di un computer. Nel caso dei file-archivio l'operazione di apertura serve appunto per creare un buffer di comunicazione tra memoria centrale e floppy.
Ogni buffer sottrae memoria, quindi quando non serve e' meglio chiuderlo.
Col Basic caricato in modo normale e' possibile gestire contemporaneamente fino a tre buffer; si puo' comunque arrivare fino ad un massimo di 15, come vedremo parlando del comando BASIC del DOS.

FILE SEQUENZIALI (prima parte)

Un file sequenziale puo' essere letto o scritto solo partendo dall'inizio e passando in rassegna tutti i suoi record.
Per registrare o leggere un file sequenziale occorre aprirlo; tale operazione e' diversa nei due casi:

```
OPEN"O",....        (per registrare)
OPEN"I",....        (per leggere)
```

Per registrare valori numerici si usa l'istruzione PRINT# seguita dal numero del buffer utilizzato, interponendo tra un dato e l'altro un punto e virgola.
Il trattamento delle stringhe richiede un'attenzione particolare per le virgole e le virgolette, in quanto questi due tipi di caratteri sono i separatori di dati alfanumerici. Usando le virgole, non si possono gestire stringhe che le contengano, per non falsare i valori in lettura.
Se si registrano stringhe senza separarle con uno dei caratteri anzidetti, esse vengono attaccate le une alle altre e non sono piu' separabili in lettura.

L E Z I O N E n. 19

INDICE

1 - TRATTAMENTO DEI FILE SEQUENZIALI (seconda parte) -

Nella lezione precedente abbiamo esaminato il trattamento delle stringhe nei file sequenziali; per poter essere lette correttamente, esse richiedono un carattere separatore. Dei due possibili, abbiamo esaminato il primo, ossia la virgola; parliamo ora del secondo: il carattere delle virgolette.

Il motivo principale per cui esiste questa duplice possibilita' di separazione delle stringhe lo abbiamo gia' detto nella lezione scorsa: se la stringa trattata contiene una o piu' virgole, l'organizzazione dei dati ne viene completamente falsata, in quanto quelle virgole diventano, in lettura, caratteri separatori e vanno cosi' ad alterare il contenuto originario delle varie stringhe.

Quindi, in presenza di questi casi, e' possibile aggirare l'ostacolo impiegando, al posto del carattere separatore virgola, quello delle virgolette.
Riprendiamo l'esempio (6) a pagina 6 della lezione precedente, e modifichiamolo in questi due modi, dal funzionamento equivalente:

```
(1)     OPEN"O",3,"SEQ4"
        PRINT#3,CHR$(34);"ROSSI MARIO";CHR$(34);CHR$(34);"VIA LARGA, 173 -
           MI";CHR$(34);
        CLOSE
                    ("ROSSI MARIO""VIA LARGA, 173 - MI")

(2)     S$=CHR$(34)
        OPEN"O",2,"SEQ5"
        PRINT#2,S$;"ROSSI MARIO";S$;S$;"VIA LARGA, 173 - MI";S$;
        CLOSE
                    ("ROSSI MARIO""VIA LARGA, 173 - MI")
```

In entrambi i casi ogni stringa registrata e' preceduta e seguita dal carattere delle virgolette, corrispondenti al codice ASCII 34. Il secondo esempio differisce dal primo per il fatto di aver depositato nella variabile alfanumerica S$ il carattere separatore; le parentesi mostrano cio' che risulta scritto sul disco.

Facciamo sull'archivio SEQ4 un'operazione di lettura dei dati nel modo solito:

```
(3)     OPEN O I",1,"SEQ4"
        INPUT#1,A$,B$
        CLOSE
        PRINT A$ : PRINT B$
```

Otteniamo queste due stringhe:

```
(4)     ROSSI MARIO                     (A$)
        VIA LARGA, 173 - MI             (B$)
```

Confrontando la (4) con la (9) a pagina 7 della lezione precedente, si nota che questa volta il trattamento delle stringhe e' corretto: la seconda viene formata come voluto, con la virgola e tutto il resto dell'indirizzo.
La lettura del file SEQ5 fornisce un risultato identico, dal momento che la (1) e la (2) registrano i rispettivi file in modo perfettamente equivalente.

Esaminiamo il seguente esempio:

```
(5)       S$=CHR$(34)
          OPEN"O",1,"SEQ6"
          PRINT#1,S$;"ROSSI MARIO";S$;"VIA LARGA, 173 - MI";S$;
          CLOSE#1
                    ("ROSSI MARIO"VIA LARGA, 173 - MI")
```

Una lettura di questo file fornisce le seguenti due stringhe:

```
(6)       ROSSI MARIO
          VIA LARGA"
```

Abbiamo nuovamente ottenuto un risultato errato. La spiegazione di questo comportamento, e di tutti gli esempi fatti e che faremo, risiede nella seguente regola:

IN FASE DI LETTURA DI UN FILE SEQUENZIALE, LE STRINGHE VENGONO FORMATE AUTOMATICAMENTE DAL COMPUTER FACENDO RIFERIMENTO AI CARATTERI SEPARATORI USATI IN REGISTRAZIONE. SE IL PRIMO CARATTERE INCONTRATO NELLA FORMAZIONE DI UNA STRINGA E' QUELLO DELLE VIRGOLETTE, NELLA STRINGA VANNO A FINIRE TUTTI I CARATTERI COMPRESI TRA TALI VIRGOLETTE E QUELLE SUCCESSIVE, SPAZI INIZIALI E VIRGOLE COMPRESE.
SE INVECE IL PRIMO CARATTERE NON E' QUELLO DELLE VIRGOLETTE, GLI EVENTUALI SPAZI INIZIALI VENGONO IGNORATI, E LA STRINGA RISULTA FORMATA DA TUTTI I CARATTERI CHE SI INCONTRANO FINO ALLA VIRGOLA SUCCESSIVA, VIRGOLETTE COMPRESE.

Come vedi, l'uso del separatore virgola differisce da quello delle virgolette anche per cio' che riguarda i caratteri bianchi (BLANKS) iniziali (per quelli intermedi e finali non ci sono differenze).

L'intera gestione delle stringhe nei file sequenziali dipende, in definitiva, dal fatto che all'interno di esse siano contenute virgole o virgolette. La presenza di uno di quei caratteri impone di usare l'altro come separatore; ne deriva, naturalmente, che una stringa contenente sia virgole che virgolette non puo' essere registrata e letta correttamente: bisogna spezzarla in due o piu' parti da trattare separatamente.

Se nessuno dei caratteri separatori compare nei dati da registrare, l'uso della virgola o delle virgolette non comporta differenze; e' evidente, comunque, che impiegando le virgole come separatori si risparmiano caratteri e battute, dal momento che una virgola fa la funzione di due virgolette.

Facciamo ora diversi esempi; tralasceremo le operazioni di apertura e chiusura dei file, per motivi di chiarezza e concisione. Come separatori useremo sia virgole che virgolette, definite rispettivamente con V$ ed S$.
Per interpretare correttamente gli esercizi tieni presente che OGNI ESEMPIO PORTA TRA LE PRIME PARENTESI QUELLO CHE VIENE REGISTRATO SU DISCO, E TRA LE RIMANENTI IL CONTENUTO CHE VA A FINIRE NELLE PRIME DUE STRINGHE LETTE.
Se hai seguito attentamente tutte le spiegazioni date finora sui file sequenziali, non dovresti trovare troppe difficolta' ad interpretare i risultati di ogni

caso prospettato.
Cerca di fugare eventuali perplessita' rileggendo attentamente la regola generale che abbiamo visto poco sopra.

```
            V$=","
            S$=CHR$(34)


(7)         PRINT#1,"NUOVA ELETTRONICA";V$;"VIA CRACOVIA, 19 - BO"
            (NUOVA ELETTRONICA,VIA CRACOVIA, 19 - BO)
               (NUOVA ELETTRONICA)
               (VIA CRACOVIA)

(8)         PRINT#2,"NUOVA ELETTRONICA";S$;"VIA CRACOVIA, 19 - BO"
            (NUOVA ELETTRONICA"VIA CRACOVIA, 19 - BO)
               (NUOVA ELETTRONICA"VIA CRACOVIA)
               (19 - BO)

(9)         PRINT#3,S$;"NUOVA ELETTRONICA";S$;S$;"VIA CRACOVIA, 19 - BO";S$
            ("NUOVA ELETTRONICA""VIA CRACOVIA, 19 - BO")
               (NUOVA ELETTRONICA)
               (VIA CRACOVIA, 19 - BO)

(10)        PRINT#1,"     NOME";V$;"  COGNOME"
            (     NOME,  COGNOME)
               (NOME)
               (COGNOME)

(11)        PRINT#2,S$;"     NOME";S$;S$;"  COGNOME";S$
            ("     NOME""  COGNOME")
               (     NOME)
               (  COGNOME)

(12)        PRINT#1,"TITOLO: ";S$;"Odissea";S$;V$;"AUTORE: Omero"
            (TITOLO: "Odissea",AUTORE: Omero)
               (TITOLO: "Odissea")
               (AUTORE: Omero)

(13)        PRINT#2,S$;"TITOLO: ";S$;"Odissea";S$;"AUTORE: Omero";S$
            ("TITOLO: "Odissea"AUTORE: Omero")
               (TITOLO: )
               (Odissea"AUTORE: Omero")

(14)        PRINT#3,S$;"TITOLO: ";S$;S$;"Odissea";S$;S$;"AUTORE: Omero";S$
            ("TITOLO: ""Odissea""AUTORE: Omero")
               (TITOLO: )
               (Odissea)

(15)        PRINT#1,"1,2";V$;"3,4"
            (1,2,3,4)
               (1)
               (2)
```

```
(16)      PRINT#2,S$;"1,2";S$;S$;"3,4";S$
          ("1,2""3,4")
             (1,2)
             (3,4)

(17)      PRINT#3,"1,2";"3,4"
          (1,23,4)
             (1)
             (23)

(18)      AA$="PRIMA STRINGA,"
          BB$="SECONDA STRINGA"
          PRINT#1,AA$;BB$
          (PRIMA STRINGA,SECONDA STRINGA)
             (PRIMA STRINGA)
             (SECONDA STRINGA)

(19)      PRINT#2,AA$;V$;BB$
          (PRIMA STRINGA,,SECONDA STRINGA)
             (PRIMA STRINGA)
             ()

(20)      PRINT#3,S$;AA$;S$;S$;BB$;S$
          ("PRIMA STRINGA,""SECONDA STRINGA")
             (PRIMA STRINGA,)
             (SECONDA STRINGA)

(21)      PRINT#1,56.1;-2*7
          ( 56.1 -14)
             ( 56.1)
             (-14 )

(22)      PRINT#2,56,1;-2*7
          ( 56              1 -14)
             ( 56 )
             ( 1 )
```

Negli ultimi due esempi i dati vengono registrati come numeri, e pertanto vanno letti con un'istruzione del tipo:

```
(23)      INPUT#1,A,B
```

Qualora si leggesse l'archivio introitando stringhe (INPUT#1,A$,B$), si otterrebbe la segnalazione d'errore (10) vista nella lezione precedente (INPUT PAST END).

Probabilmente, se arrivi qui dopo aver esaminato attentamente tutti gli esempi, ti fuma la testa!
E' comunque estremamente importante capire bene il meccanismo di funzionamento dei file sequenziali; rileggi quindi quei casi che non ti sono troppo chiari. Per ognuno di essi sono state date diverse versioni; confrontandole tra di loro dovresti arrivare a capirle tutte.

Non ci dilungheremo in complesse spiegazioni degli esempi fatti: occorrerebbero parecchie pagine e faremmo fatica a farci intendere. Del resto le combinazioni possibili sono pressoche' illimitate e sarebbe un'impresa improba cercare di vederle tutte. Solo l'esperienza diretta potra' aiutarti ad assimilare questi concetti.

Nella prossima lezione dovrai scrivere un programma dove compare la gestione di un file sequenziale; avrai quindi la possibilita' di fare le prime esperienze in modo guidato: poco per volta ogni elemento andra' al posto giusto e la soddisfazione che ne trarrai sara' proporzionale alla fatica impiegata.
Parleremo anche dei comandi LIST e PRINT del DOS, per mezzo dei quali e' possibile verificare il reale contenuto di un file, ossia cio' che risulta registrato sul dischetto; il loro uso ti chiarira' definitivamente le idee.

ESERCIZIO n. 1

Si abbia il seguente programma:

```
100 OPEN"O",1,"PROVA"
110 PRINT#1,1;"- NOME";",";2;"- COGNOME"
120 CLOSE
```

Cosa viene scritto sul disco nell'archivio PROVA?
.....
.....
.....
.....

RISPOSTA

Il file PROVA ha il seguente contenuto:

```
( 1 - NOME, 2 - COGNOME)
```

Abbiamo messo le parentesi per evidenziare che il file inizia con un blank: infatti le cifre 1 e 2 vengono registrate come numeri, quindi sono precedute e seguite da uno spazio.

ESERCIZIO n. 2

Prendiamo in esame ancora il programma dell'esercizio precedente. Cosa si ottiene con la seguente lettura?

```
130 OPEN"I",1,"PROVA"
140 INPUT#1,A$
..........
..........
..........
..........
```

RISPOSTA

Nella variabile alfanumerica A$ va a finire la seguente stringa:

```
(1 - NOME)
```

(con la solita considerazione sulle parentesi).
Da notare che il numero 1 e' stato letto assieme al resto: richiedendo una stringa, in essa viene depositato tutto quello che si trova davanti alla prima virgola registrata nel file. Cio' spiega il fatto che si e' perso il blank iniziale, che si ha solo se la cifra 1 viene letta come numero e non come stringa.
Evidentemente, allora, il file in oggetto puo' essere letto in vari modi; se seguiamo in lettura lo stesso criterio adottato per la registrazione, si ha:

```
130 OPEN"I",1,"PROVA"
140 INPUT#1,A,A$,B,B$
150 CLOSE
```

Nelle variabili numeriche A e B vanno rispettivamente i numeri 1 e 2, mentre in A$ e B$ si ottengono le diciture "- NOME" e "- COGNOME".

---

2 - ESPANSIONE DI UN FILE SEQUENZIALE -

Quando si effettua un'apertura del tipo:

```
(24)      OPEN"O",1,"PROVA"
```

il file specificato viene sempre registrato dall'inizio: cio' significa che se esso esisteva gia', tutti i dati precedenti vengono cancellati e sostituiti con quelli nuovi.

Bisogna avere ben chiaro questo comportamento; la naturale conseguenza e' che se dobbiamo cambiare il sesto dato in un archivio sequenziale che ne contenga, ad esempio, 359 in tutto, bisogna riinciderlo dall'inizio, un record dopo l'altro.

Nella stragrande maggioranza dei personal computer bisogna sottostare alla medesima regola anche per espandere un file: si deve registrarlo da capo, aggiungendo alla fine i nuovi valori.
Nel computer di NUOVA ELETTRONICA esiste invece una grande comodita': e' possibile registrare dati in coda ad un file sequenziale preesistente, senza doverlo riscrivere tutto; i nuovi record vengono semplicemente appesi a quelli precedenti.

L'istruzione che permette questa prestazione e' la seguente:

```
(25)      OPEN"E",1,"PROVA"
```

Con una apertura di questo tipo il precedente contenuto del file PROVA non viene toccato; ogni istruzione PRINT# attacca i nuovi dati in coda a quelli preesistenti.

Quando si crea un nuovo file sequenziale, l'apertura (25) puo' essere usata anche al posto di quella normale (24).
Questa procedura non manda in errore il calcolatore; ovviamente i dati vengono registrati partendo dall'inizio del nuovo archivio.

Naturalmente se si desidera variare un dato intermedio occorre sempre riscrivere l'intero archivio effettuandone l'apertura con la (24).

3 - SEGNALAZIONI D'ERRORE -

Nel trattamento dei file si possono commettere svariati errori, sia nella scrittura delle istruzioni, sia nella gestione dei dati; non e' quindi pensabile fare un elenco di tutte le possibili segnalazioni d'errore, spiegandole poi in dettaglio. Ci limiteremo allora ad indicare le piu' frequenti, rimandando all'appendice 2 dell'introduzione per la relativa spiegazione.

```
(26)      BAD FILE DATA
          INTERNAL ERROR
          BAD FILE NUMBER
          FILE NOT FOUND
          BAD FILE MODE
          FILE ALREADY OPEN
          DISK I/O ERROR
          DISK FULL
          INPUT PAST END
          BAD RECORD NUMBER
          BAD FILE NAME
          TOO MANY FILES
```

Non ti sara' difficile, caso per caso, capire qual'e' stato l'errore commesso.

Il numero massimo di file (tra programmi ed archivi) che e' possibile mettere in un dischetto e' 48; superando tale limite si ha l'ultima segnalazione d'errore.
Consideriamo la seguente istruzione:

```
(27)      OPEN"O",1,"PROVA:2"
```

Con essa si ordina al computer di andare sul drive 2 a registrare il file PROVA; se lo sportello del drive non e' chiuso si ha sempre l'ultima delle segnalazioni precedenti.

La seconda delle (26) si ha quando si cerca di registrare un file su di un disco che porta il nastro di protezione contro le registrazioni.

Ancora una raccomandazione: occorre prestare molta attenzione ai nomi usati per gli archivi, siano essi sequenziali o random; se il nome dato ad un file corrisponde a quello di un programma gia' presente su disco, esso viene cancellato e al suo posto troveremo l'archivio. C'e' quindi il pericolo di perdere inavvertitamente programmi preziosi. Tale evenienza puo' essere scongiurata facendo sempre una DIR sui dischi che si usano, e tenendo regolarmente una copia dei programmi e degli archivi importanti.

Dopo la stampa del riepilogo, esercitati nella gestione dei file sequenziali; la procedura che potresti seguire e' quella di eseguire realmente gli esempi che vanno dal (7) al (22), usando allo scopo un programma di questo tipo:

```
10 V$=","  : S$=CHR$(34)
20 OPEN"O",1,"SEQ1"
30 PRINT#1, .....
40 CLOSE
50 OPEN"I",1,"SEQ1"
60 INPUT#1,A$,B$
70 CLOSE
80 PRINT A$ : PRINT B$
```

Al posto dei puntini metterai volta a volta le stringhe giuste. Dopo aver controllato di aver digitato tutto in modo corretto, puoi dare il RUN; naturalmente il disco su cui farai le prove non deve avere il nastro di protezione. Il computer eseguira' il suo lavoro e sul monitor vedrai i valori assunti da A$ e B$.
Negli esempi (21) e (22) dovrai mettere, nella righe 60 e 80 del programma, A e B al posto di A$ e B$.
Se esegui tutto con precisione otterrai per ogni esercizio le diciture poste tra le ultime due parentesi.

RIEPILOGO DELLA LEZIONE 19

FILE SEQUENZIALI (seconda parte)

Oltre alla virgola, anche le virgolette sono un carattere separatore di stringhe nella gestione dei file sequenziali. Esse non possono essere usate per trattare stringhe che le contengono (come accade per il carattere separatore virgola).
Se il carattere iniziale e' quello delle virgolette, nella stringa letta vanno a finire tutti i caratteri successivi, fino ad arrivare alle virgolette seguenti; entrano a far parte della stringa anche eventuali spazi iniziali e virgole.
Se il primo carattere letto e' diverso dalle virgolette, la stringa e' formata da tutto cio' che si trova fino alla prima virgola, virgolette comprese; gli eventuali spazi iniziali vengono ignorati.

ESPANSIONE DEI FILE SEQUENZIALI

Un file sequenziale puo' essere aperto in registrazione anche per appendere dati in coda ai precedenti; l'istruzione relativa e' di questo tipo:

```
OPEN"E",....
```

In ogni caso, pero', per cambiare un valore intermedio in un archivio sequenziale occorre riscriverlo completamente.

L E Z I O N E n. 20

INDICE

1 - ONERRORGOTO - (= IN CASO D'ERRORE VAI A)

L'istruzione ONERRORGOTO riveste una grande importanza, in quanto permette, se usata in modo appropriato, di gestire gli errori che possono verificarsi durante lo svolgimento di un programma.
L'argomento e' molto vasto, ed ora lo introduciamo solo perche' ci serve nella stesura del programma che devi scrivere tra poco; per il momento, quindi, faremo solo qualche cenno.

L'istruzione in oggetto (che puo' essere scritta anche ON ERROR GOTO, ossia separando le tre parole componenti) ordina al computer di andare al numero di riga specificato qualora, per qualsiasi motivo, si verifichi un errore.
Alla linea indicata inizia quella che viene chiamata comunemente ROUTINE DI CORREZIONE DEGLI ERRORI: in essa, su una o piu' righe, vengono analizzate le varie possibilita' d'errore; per ognuna si prevede un comportamento opportuno, in modo da rientrare nel programma senza danni.

Supponiamo, ad esempio, che in una certa linea venga richiesto un dato numerico con l'istruzione INPUT; se tale valore viene successivamente usato al denominatore di una frazione, si incorre in errore col valore zero, perche' non e' possibile dividere un numero qualsiasi per zero. In un caso simile l'esecuzione del programma viene interrotta e sul monitor compare la seguente segnalazione d'errore:

```
(1)       DIVISION BY ZERO          (divisione per zero)
```

Per utilizzare una routine di gestione degli errori occorre mettere (generalmente all'inizio del programma) l'istruzione ONERRORGOTO, seguita da un numero di linea:

```
(2)       20 ONERRORGOTO 800
```

Nell'esempio, alla riga 800 inizia la routine di correzione.
In pratica questa istruzione viene memorizzata dal computer, che in caso d'errore (in qualsiasi punto del programma abbia luogo) effettua un salto incondizionato alla linea specificata.

Per gestire i vari tipi di errore esistono poi altre quattro parole chiave:

```
(3)       ERROR
          ERL
          ERR
          RESUME
```

Per ora accenneremo solo all'uso di RESUME.

2 - RESUME - (= RICOMINCIA DA)

Ogni routine di correzione d'errore deve terminare con un'istruzione RESUME; essa serve per far rientrare il programma nel punto desiderato. Infatti RESUME deve essere seguito dal numero di linea in cui va ripresa l'elaborazione dei dati.

Esistono pero' due alternative: scrivere RESUME da solo o mettere NEXT la posto del rinvio. Quindi RESUME puo' essere dato in uno dei seguenti modi:

```
(4)     RESUME 230
        RESUME
        RESUME NEXT
```

Nel primo esempio il programma, in caso d'errore, riprende dalla linea 230, mentre nel secondo il rientro avviene nella riga in cui si e' verificato l'errore; nell'ultimo caso si va alla riga successiva a quella d'errore (NEXT=successivo). In effetti una routine di correzione e' piu' complessa: generalmente si usano le parole chiave ERL e ERR (numero di linea e numero di codice dell'errore, rispettivamente); per ora ci fermeremo a questo punto.

A titolo d'esempio, vediamo come si potrebbe gestire un errore derivante da una divisione per zero:

```
(5)     100 ONERRORGOTO 800
        .....
        230 INPUT G3
        240 D#=A/G3
        .....
        800 PRINT "divisione per zero !"
        810 RESUME 230
```

Alla linea 230 si deve introdurre il valore da assegnare a G3; in quella successiva G3 compare al denominatore di una frazione. Se si attribuisce a quella variabile numerica il valore zero e se non esiste una routine di correzione dell'errore, si esce dal programma. Nel nostro caso, invece, si passa alla linea 800, dove c'e' l'istruzione di visualizzare la frase tra virgolette; alla riga 810 troviamo RESUME 230, che fa rientrare al punto voluto.

Avremo occasione di vedere vari esempi pratici sull'uso delle istruzioni che ora abbiamo appena accennato. Il primo esempio applicativo si trova nel programma che segue.

3 - SCRITTURA DI UN PROGRAMMA -

Tra poco devi scrivere un programma, analogamente a quanto hai gia' fatto alla lezione 15; anzi, si tratta di un certo numero di linee che vanno inserite proprio in mezzo o alla fine del programma STAT1 di allora.
E' quindi opportuno che tu vada a rivedere le considerazioni fatte sul suo funzionamento, per potere seguire meglio le modifiche che apporteremo in questa sede; in particolare, faremo riferimento al listato (9) di pagina 9 della lezione 15. In quella prossima vedremo come si puo' riuscire ad unire i due spezzoni, in modo da ottenere un programma unico.

Innanzitutto dichiariamo quali sono gli obiettivi che vogliamo raggiungere.
Il programma STAT1 chiedeva l'eta' delle persone trattate, teneva il conto dei dati introdotti e calcolava le eta' minima, media e massima tra quelle digitate.
Ora vogliamo fare le seguenti aggiunte:

1 - Desideriamo registrare nome e cognome delle persone interessate, nonche' il loro indirizzo completo di telefono. Si tratta quindi di creare un archivio su disco, in modo da non perdere i dati.
2 - Tale archivio deve essere espandibile tramite registrazioni successive.
3 - Vogliamo memorizzare anche i valori dei parametri trattati da STAT1 (ossia le variabili numeriche N - MI - MA - ME%), tenendoli costantemente aggiornati mano a mano che aggiungiamo nominativi nuovi.

I punti 1 e 2 impongono la creazione di un file sequenziale espandibile, in modo che ogni registrazione di dati avvenga in coda a quelli gia' presenti in archivio. Assegneremo il nome ANAGRAF1 a questo file.
Per soddisfare alle condizioni del punto 3 basta creare un altro file (che chiameremo DATANAG) contenente solo quei quattro valori numerici; quando si effettua la chiusura di una seduta di registrazione dei dati (introducendo il numero zero in risposta all'INPUT della linea 120), tutto il contenuto del file viene aggiornato.
Ovviamente una seduta successiva ha come base di partenza gli ultimi dati registrati.

Ora possiamo incominciare; il metodo usato e' uguale a quello della lezione 15. La numerazione delle linee non procede di 10 in 10 perche' esse devono inserirsi al punto giusto tra quelle del programma STAT1. Le istruzioni usate sono le seguenti:

```
CLEAR
CLS
PRINT
ONERRORGOTO
OPEN"I"
INPUT#
CLOSE
OPEN"E"
INPUT
OPEN"O"
PRINT#
END
RESUME
```

Se qualcuna di queste non ti e' troppo chiara, e' meglio che tu vada subito a riguardartela.

Se non riesci a scrivere correttamente il contenuto di una linea di programma, puoi proseguire digitando in sua vece un punto interrogativo. Sul monitor comparira' la linea esatta.

LINEA 101

Questa riga serve per riservare alle stringhe 150 caratteri, in luogo dei 50 soliti.

LINEA 102

Cancellazione dello schermo.

LINEA 103

Il contenuto di questa riga deve far scrivere sul monitor la frase ATTENDERE PREGO. In questo, come in tutti i casi seguenti di stampa, puoi usare al posto della parola chiave PRINT la sua abbreviazione.

LINEA 104

In caso di errore si deve passare alla linea 400.

LINEA 105

Istruzione per aprire in lettura il file sequenziale DATANAG, utilizzando il buffer numero 1.

LINEA 106

Dal file appena aperto si debbono leggere 4 numeri, depositandoli nelle seguenti variabili: N - MI - MA - ME%.

LINEA 107

Questa riga deve contenere l'istruzione per effettuare la chiusura del file che utilizza il buffer numero 1.

LINEA 108

Viene aperto il file sequenziale ANAGRAF1, con possibilita' di registrazioni in coda ai dati precedenti; si utilizza il buffer 1.

LINEA 109

La variabile di stringa V$ deve contenere il carattere della virgola; essa va definita in modo diretto.

LINEA 142

In questa riga deve essere introitata la variabile di stringa C$, con la dicitura "COGNOME".

LINEA 143

Ora deve essere richiesta la variabile di stringa N$, con la dicitura "NOME".

LINEA 144

Qui viene inputata la variabile di stringa I$, con la dicitura "VIA".

LINEA 145

Viene introitata la variabile di stringa S$, con la dicitura "CITTA'".

LINEA 146

In questa linea viene richiesta la variabile di stringa T$, con la dicitura "TELEF.".

LINEA 311

Deve contenere l'istruzione che fa registrare nel file 1, una di seguito all'altra, queste variabili: C$ - V$ - N$ - V$ - E% - I$ - V$ - S$ - V$ - T$ - V$.
Le registrazioni successive devono avvenire senza perdita di spazio sul disco.

LINEA 312

Questa e' una riga ad istruzioni multiple e deve azzerare il contenuto delle variabili C$ - N$ - I$ - S$ - T$.

LINEA 340

Cancellazione dello schermo.

LINEA 350

L'istruzione deve visualizzare la dicitura "CHIUSURA DEL LAVORO - ATTENDERE".

LINEA 360

Questa riga deve contenere l'apertura del file DATANAG in registrazione normale (ossia dall'inizio dell'archivio); deve essere utilizzato il buffer numero 2.

LINEA 370

Nel file DATANAG bisogna registrare, uno di seguito all'altro, i valori delle variabili numeriche N - MI - MA - ME%.

LINEA 380

Istruzione di chiusura di tutti i file aperti.

LINEA 390

Questa riga deve contenere l'istruzione che pone termine al programma.

LINEA 400

Siamo all'ultima linea; essa contiene la routine di correzione degli errori. Bisogna scrivere l'istruzione che fa rientrare nella riga 107.

4 - SPIEGAZIONE DEL PROGRAMMA (prima parte) -

Il programma che hai appena finito di scrivere si trova ora registrato sul disco col nome STAT2 (per distinguerlo dal precedente STAT1).

Come abbiamo gia' detto, esso contiene tutte le aggiunte e le modifiche che era necessario apportare al primo programma per raggiungere gli obiettivi dichiarati; nella prossima lezione vedremo come e' possibile fondere STAT1 con STAT2, in modo da ottenerne uno solo.

In questa lezione ci limiteremo a fornire i listati dei due programmi STAT2:

quello proposto da noi e quello scritto da te. Cominciamo dal primo:

(6)

```
101 CLEAR 150
102 CLS
103 PRINT "ATTENDERE PREGO"
104 ONERRORGOTO 400
105 OPEN"I",1,"DATANAG"
106 INPUT#1,N,MI,MA,E%
107 CLOSE#1
108 OPEN"E",1,"ANAGRAF1"
109 V$=","
142 INPUT "COGNOME ";C$
143 INPUT "NOME    ";N$
144 INPUT "VIA     ";I$
145 INPUT "CITTA'  ";S$
146 INPUT "TELEF.  ";T$
311 PRINT#1,C$;V$;N$;E%;I$;V$;S$;V$;T$;V$;
312 C$="" : N$="" : I$="" : S$="" : T$=""
340 CLS
350 PRINT "CHIUSURA DEL LAVORO - ATTENDERE"
360 OPEN"O",2,"DATANAG"
370 PRINT#2,N;MI;MA;ME%
380 CLOSE
390 END
400 RESUME 107
```

Vediamo ora la versione che hai digitato:

(7)

```
101 CLEAR 150
102 CLS
103 PRINT "ATTENDERE PREGO"
104 ONERRORGOTO 400
105 OPEN "I",1,"DATANAG"
106 INPUT#1,N,MI,MA,E%
107 CLOSE#1
108 OPEN"E",1,"ANAGRAF1"
109 V$=","
142 INPUT"COGNOME";C$
143 INPUT"NOME";N$
144 INPUT"VIA";I$
145 INPUT"CITTA'";S$
146 INPUT"TELEF.";T$
311 PRINT#1,C$;V$;N$;V$;E%;I$;V$;S$;V$;T$;V$;
312 C$="" : N$="" :I$="":S$="":T$=""
340 CLS
350 PRINT"CHIUSURA DEL LAVORO - ATTENDERE"
360 OPEN"O",2,"DATANAG"
370 PRINT#2,N;MI;MA;ME%
380 CLOSE
390 END
400 RESUME 107
```

Molto probabilmente ci sono alcune differenze, come una diversa disposizione degli spazi o alcuni caratteri scritti in minuscolo invece che in maiuscolo; in ogni caso, pero', esso risulta formalmente e sintatticamente corretto.

Lo spazio che avevamo a disposizione per questa lezione e' terminato; vedremo quindi nella prossima la spiegazione delle linee piu' significative.
Per adesso puoi fare un confronto approfondito delle due versioni del programma STAT2; rammentiamo ancora una volta che tutte le istruzioni che hai digitato in caratteri minuscoli appariranno poi in maiuscolo nel vero listato (ad eccezione di quanto e' compreso tra le virgolette).

RIEPILOGO DELLA LEZIONE 20

ONERRORGOTO

L'istruzione ONERRORGOTO serve per rinviare il programma alla routine che gestisce gli errori; viene normalmente posta all'inizio del programma.
Non si tratta, ovviamente, degli errori di programmazione, ma di quelli che avvengono per un cattivo uso del programma da parte dell'utilizzatore (od anche per altri motivi, come vedremo).
La gestione dei vari tipi d'errore viene fatta con le istruzioni ERR, ERL e ERROR.

RESUME

Ogni routine di trattamento degli errori deve terminare con un'istruzione di rientro nella linea principale del programma; tale funzione e' svolta da RESUME. Il ritorno puo' avvenire nella linea in cui si e' verificato l'errore, in quella successiva, oppure in una specificata. Le tre forme ammesse sono quindi:

```
RESUME
RESUME NEXT
RESUME 3000
```