Programmare in C su Amiga

di Dario de Judicibus

Ultima puntata dedicata alle mappe di tastiera. Con questa puntata scopriremo tutti i segreti della mappa di tastiera italiana. Finiremo inoltre la scheda tecnica alla gadtools.library. *Una puntata un po' più lunga del solito, dato che si tratta proprio dell'ultima puntata in assoluto, ed abbiamo voluto chiudere il discorso senza lasciare le cose a metà. Non preoccupatevi comunque. Ci occuperemo ancora di Amiga e di programmazione nei prossimi numeri (e di molto altro ancora), con una serie di articoli rinnovati nello stile e negli obiettivi. Ma di questo parleremo più avanti*

La tastiera italiana

Nella scorsa puntata abbiamo iniziato l'analisi esadecimale del file **keymaps/i** che contiene la definizione della mappa di tastiera *italiana*.

Abbiamo visto che questo file è composto da un *hunk_header* di sei parole (quattro byte l'una), un *hunk_code* di ben 282 parole, un *hunk_reloc32* formato da 43 parole ed in fondo l'*hunk_end*, formato solo dal suo codice di identificazione (una parola).

La definizione della tastiera si trova nell'*hunk_code* centrale.

In questa puntata analizzeremo in dettaglio questo *hunk*, in modo da estrarne le caratteristiche della tastiera italiana, e confrontarle con l'analisi teorica esposta nelle scorse puntate.

Innanzi tutto, dato che tutti i puntatori contenuti in questo *hunk* sono relativi alla posizione del primo byte del corpo vero e proprio dell'*hunk*, ho riportato in figura 1 il contenuto del file **keymaps/i** in formato esadecimale, con a fianco una scala di indirizzi opportunamente spostata di 32 posizioni (**0x20**). Le due parole poste *prima* del nuovo indirizzo **0000** sono quindi il codice di identificazione dell'*hunk_code* (**0x03E9**) ed il numero di parole del blocco dati (**0x11A**, cioè **282** appunto).

Consideriamo ora la struttura riportata in figura 2. Si tratta in pratica di una struttura **Node** (vista nelle prime puntate di questa rubrica), contenente in fondo una struttura **KeyMap** (riportata in figura 3). Proviamo ad associare questa struttura alle prime parole del blocco che stiamo analizzando. Il risultato, riportato in figura 4, mostra che in effetti la prima parte del blocco in questione rappresenta proprio la struttura or ora definita.

In particolare, il campo **ln_Name** contiene il puntatore al nome della tastiera che, nel nostro caso, è una stringa di un carattere (la **"i"**). Si tratta di una tipica stringa **C**, detta anche *a chiusura nulla* [*null-terminated*], in quanto termina con un byte nullo.

Osservate ora la prima colonna della seconda tabella in figura 4, e precisamente le righe comprese tra la quinta riga e la tredicesima incluse. Questa colonna contiene gli indirizzi delle parole il cui valore si trova nella seconda colonna. Quelle facenti parte del gruppo selezionato contengono *tutte* un puntatore ad un'altra parola dello stesso blocco dati. Se ora andate a vedere le ultime parole dell'*hunk_reloc32*, vi troverete gli stessi indirizzi, riportati in ordine decrescente. Questo vuol dire che i puntatori in questione verranno *rilocati* quando il blocco sarà caricato in memoria.

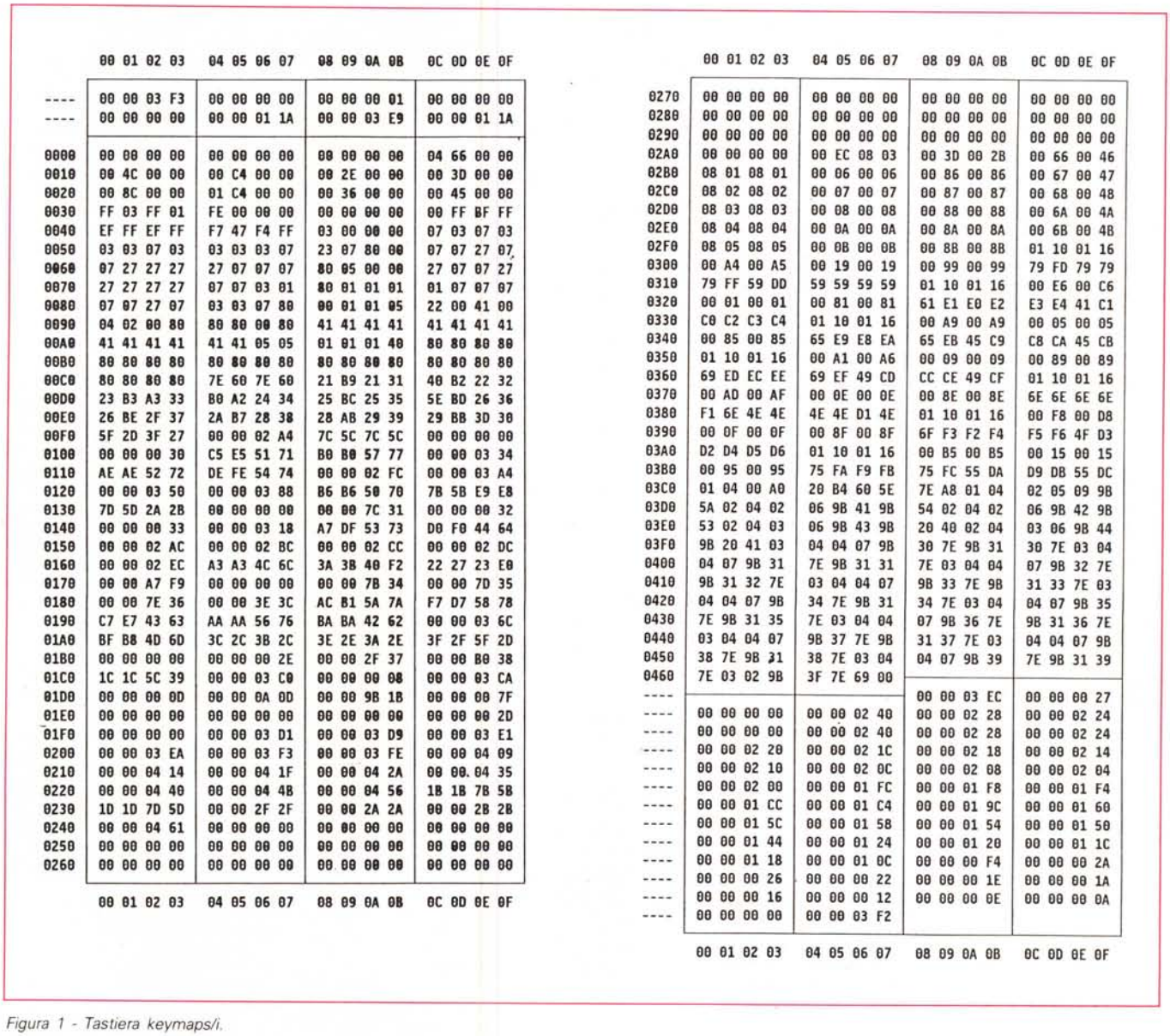
Veniamo ora alle varie tabelle della mappa di tastiera.

km_LoCapsable e km_HiCapsable

In figura 5 e in figura 6 sono riportate le due tabelle contenenti le informazioni relative all'effetto del blocco del maiuscolo sui singoli tasti. In ogni figura sono riportate due tabelle. La prima mostra la posizione del blocco nella mappa di tastiera, la seconda mostra lo stesso blocco esploso in modo da riportarlo sia in formato esadecimale, che in formato binario. Questa tabella inoltre riporta i singoli codici di scansione con accanto, tra parentesi, **"S"** se il tasto è influenzato dal blocco del maiuscolo, **"N"** se non lo è.

Ad esempio, il bit 0 del terzo byte nella tabella relativa alla parte bassa della mappa di tastiera, quella cioè riportata in figura 5, è **"1"**. Questo vuol dire che quando è attivo il blocco del maiuscolo, il tasto corrispondente al codice di scansione **0x10** emette lo stesso carattere che emetterebbe se fosse premuto insieme al modificatore *shift*, in condizioni normali. Ed in effetti i conti tornano, visto che si tratta della lettera **"q"**.

Viceversa, il bit 1 del primo byte nella stessa tabella, vale **"0"**. Questo vuol dire che il tasto associatovi non è influenzato dal blocco del maiuscolo. Nella tastiera italiana, questo tasto corrisponde a quello marcato da un *uno* e dal *punto esclamativo*, il quale infatti



	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
----	00 00 03 F3	00 00 00 00	00 00 00 01	00 00 00 00
----	00 00 00 00	00 00 01 1A	00 00 03 E9	00 00 01 1A
0000	00 00 00 00	00 00 00 00	00 00 00 00	04 66 00 00
0010	00 4C 00 00	00 C4 00 00	00 2E 00 00	00 3D 00 00
0020	00 8C 00 00	01 C4 00 00	00 36 00 00	00 45 00 00
0030	FF 03 FF 01	FE 00 00 00	00 00 00 00	00 FF BF FF
0040	EF FF EF FF	F7 47 F4 FF	03 00 00 00	07 03 07 03
0050	03 03 07 03	03 03 03 07	23 07 80 00	07 07 27 07
0060	07 27 27 27	27 07 07 07	80 05 00 00	27 07 07 27
0070	27 27 27 27	07 07 03 01	80 01 01 01	01 07 07 07
0080	07 07 27 07	03 03 07 80	00 01 01 05	22 00 41 00
0090	04 02 00 80	80 80 00 80	41 41 41 41	41 41 41 41
00A0	41 41 41 41	41 41 05 05	01 01 01 40	80 80 80 80
00B0	80 80 80 80	80 80 80 80	80 80 80 80	80 80 80 80
00C0	80 80 80 80	7E 60 7E 60	21 B9 21 31	40 B2 22 32
00D0	23 B3 A3 33	B0 A2 24 34	25 BC 25 35	5E BD 26 36
00E0	26 BE 2F 37	2A B7 28 38	28 AB 29 39	29 BB 3D 30
00F0	5F 2D 3F 27	00 00 02 A4	7C 5C 7C 5C	00 00 00 00
0100	00 00 00 30	C5 E5 51 71	B0 B0 57 77	00 00 03 34
0110	AE AE 52 72	DE FE 54 74	00 00 02 FC	00 00 03 A4
0120	00 00 03 50	00 00 03 88	B6 B6 50 70	7B 5B E9 E8
0130	7D 5D 2A 2B	00 00 00 00	00 00 7C 31	00 00 00 32
0140	00 00 00 33	00 00 03 18	A7 DF 53 73	D0 F0 44 64
0150	00 00 02 AC	00 00 02 BC	00 00 02 CC	00 00 02 DC
0160	00 00 02 EC	A3 A3 4C 6C	3A 3B 40 F2	22 27 23 E0
0170	00 00 A7 F9	00 00 00 00	00 00 7B 34	00 00 7D 35
0180	00 00 7E 36	00 00 3E 3C	AC B1 5A 7A	F7 D7 58 78
0190	C7 E7 43 63	AA AA 56 76	BA BA 42 62	00 00 03 6C
01A0	BF B8 4D 6D	3C 2C 3B 2C	3E 2E 3A 2E	3F 2F 5F 2D
01B0	00 00 00 00	00 00 00 2E	00 00 2F 37	00 00 B0 38
01C0	1C 1C 5C 39	00 00 03 C0	00 00 00 08	00 00 03 CA
01D0	00 00 00 0D	00 00 0A 0D	00 00 9B 1B	00 00 00 7F
01E0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 2D
01F0	00 00 00 00	00 00 03 D1	00 00 03 D9	00 00 03 E1
0200	00 00 03 EA	00 00 03 F3	00 00 03 FE	00 00 04 09
0210	00 00 04 14	00 00 04 1F	00 00 04 2A	00 00 04 35
0220	00 00 04 40	00 00 04 4B	00 00 04 56	1B 1B 7B 5B
0230	1D 1D 7D 5D	00 00 2F 2F	00 00 2A 2A	00 00 2B 2B
0240	00 00 04 61	00 00 00 00	00 00 00 00	00 00 00 00
0250	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0260	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0270	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0280	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0290	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
02A0	00 00 00 00	00 EC 08 03	00 3D 00 2B	00 66 00 46
02B0	08 01 08 01	00 06 00 06	00 86 00 86	00 67 00 47
02C0	08 02 08 02	00 07 00 07	00 87 00 87	00 68 00 48
02D0	08 03 08 03	00 08 00 08	00 88 00 88	00 6A 00 4A
02E0	08 04 08 04	00 0A 00 0A	00 8A 00 8A	00 6B 00 4B
02F0	08 05 08 05	00 0B 00 0B	00 8B 00 8B	01 10 01 16
0300	00 A4 00 A5	00 19 00 19	00 99 00 99	79 FD 79 79
0310	79 FF 59 DD	59 59 59 59	01 10 01 16	00 E6 00 C6
0320	00 01 00 01	00 81 00 81	61 E1 E0 E2	E3 E4 41 C1
0330	C0 C2 C3 C4	01 10 01 16	00 A9 00 A9	00 05 00 05
0340	00 85 00 85	65 E9 E8 EA	65 EB 45 C9	C8 CA 45 CB
0350	01 10 01 16	00 A1 00 A6	00 09 00 09	00 89 00 89
0360	69 ED EC EE	69 EF 49 CD	CC CE 49 CF	01 10 01 16
0370	00 AD 00 AF	00 0E 00 0E	00 8E 00 8E	6E 6E 6E 6E
0380	F1 6E 4E 4E	4E 4E D1 4E	01 10 01 16	00 F8 00 D8
0390	00 0F 00 0F	00 8F 00 8F	6F F3 F2 F4	F5 F6 4F D3
03A0	D2 D4 D5 D6	01 10 01 16	00 B5 00 B5	00 15 00 15
03B0	00 95 00 95	75 FA F9 FB	75 FC 55 DA	D9 DB 55 DC
03C0	01 04 00 A0	20 B4 60 5E	7E A8 01 04	02 05 09 9B
03D0	5A 02 04 02	06 9B 41 9B	54 02 04 02	06 9B 42 9B
03E0	53 02 04 03	06 9B 43 9B	20 40 02 04	03 06 9B 44
03F0	9B 20 41 03	04 04 07 9B	30 7E 9B 31	30 7E 03 04
0400	04 07 9B 31	7E 9B 31 31	7E 03 04 04	07 9B 32 7E
0410	9B 31 32 7E	03 04 04 07	9B 33 7E 9B	31 33 7E 03
0420	04 04 07 9B	34 7E 9B 31	34 7E 03 04	04 07 9B 35
0430	7E 9B 31 35	7E 03 04 04	07 9B 36 7E	9B 31 36 7E
0440	03 04 04 07	9B 37 7E 9B	31 37 7E 03	04 04 07 9B
0450	38 7E 9B 31	38 7E 03 04	04 07 9B 39	7E 9B 31 39
0460	7E 03 02 9B	3F 7E 69 00		
----			00 00 03 EC	00 00 00 27
----	00 00 00 00	00 00 02 40	00 00 02 28	00 00 02 24
----	00 00 00 00	00 00 02 40	00 00 02 28	00 00 02 24
----	00 00 02 20	00 00 02 1C	00 00 02 18	00 00 02 14
----	00 00 02 10	00 00 02 0C	00 00 02 08	00 00 02 04
----	00 00 02 00	00 00 01 FC	00 00 01 F8	00 00 01 F4
----	00 00 01 CC	00 00 01 C4	00 00 01 9C	00 00 01 60
----	00 00 01 5C	00 00 01 58	00 00 01 54	00 00 01 50
----	00 00 01 44	00 00 01 24	00 00 01 20	00 00 01 1C
----	00 00 01 18	00 00 01 0C	00 00 00 F4	00 00 00 2A
----	00 00 00 26	00 00 00 22	00 00 00 1E	00 00 00 1A
----	00 00 00 16	00 00 00 12	00 00 00 0E	00 00 00 0A
----	00 00 00 00	00 00 03 F2		

Figura 1 - Tastiera keymaps/i.

```
struct KeyMapNode
{
   struct Node   *ln_Succ ;
   struct Node   *ln_Pred ;
   UBYTE          ln_Type ;
   BYTE           ln_Pri  ;
   char          *ln_Name ;
   struct KeyMap  kmap    ;
}
```

Figura 2 - Nodo contenente una struttura KeyMap.

```
struct  KeyMap
{
  UBYTE  *km_LoKeyMapTypes ; // Base: struttura delle definizioni
  ULONG  *km_LoKeyMap      ; // Base: sequenze di emissione
  UBYTE  *km_LoCapsable    ; // Base: effetto del blocco del maiuscolo
  UBYTE  *km_LoRepeatable  ; // Base: emissione continua
  UBYTE  *km_HiKeyMapTypes ; // Speciali: struttura delle definizioni
  ULONG  *km_HiKeyMap      ; // Speciali: sequenze di emissione
  UBYTE  *km_HiCapsable    ; // Speciali: effetto del blocco del maiuscolo
  UBYTE  *km_HiRepeatable  ; // Speciali: emissione continua
};
```

Figura 3 - Struttura KeyMap.

non cambia comportamento quando è attivo il blocco del maiuscolo, permettendo così di immettere anche in quel caso il numero uno, senza bisogno di premere *shift*.

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0000	00 00 00 00	00 00 00 00	00 00 00 00	04 66 00 00
0010	00 4C 00 00	00 C4 00 00	00 2E 00 00	00 3D 00 00
0020	00 8C 00 00	01 C4 00 00	00 36 00 00	00 45

0000	00 00 00 00	Nessun successore	ln_Succ
0004	00 00 00 00	Nessun predecessore	ln_Pred
0008	00	NT_UNKNOWN	ln_Type
0009	00	Priorità 0	ln_Pri
000A	00 00 04 66	-> 69 00 = i\0 (nome)	ln_Name
000E	00 00 00 4C	-> 07 03 07 03 ...	km_LoKeyMapTypes
0012	00 00 00 C4	-> 7E 60 7E 60 ...	km_LoKeyMap
0016	00 00 00 2E	-> 00 00 FF 03 ...	km_LoCapsable
001A	00 00 00 3D	-> FF BF FF EF ...	km_LoRepeatable
001E	00 00 00 8C	-> 22 00 41 00 ...	km_HiKeyMapTypes
0022	00 00 01 C4	-> 00 00 03 C0 ...	km_HiKeyMap
0026	00 00 00 36	-> 00 00 00 00 ...	km_HiCapsable
002A	00 00 00 45	-> 47 F4 FF 03 ...	km_HiRepeatable

▲ *Figura 4*
Associazione tra KeyMapNode e la prima parte del blocco dati.

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0020				00 00
0030	FF 03 FF 01	FE 00		

002E	00	0000 0000	07(N) 06(N) 05(N) 04(N) 03(N) 02(N) 01(N) 00(N)
002F	00	0000 0000	0F(N) 0E(N) 0D(N) 0C(N) 0B(N) 0A(N) 09(N) 08(N)
0030	FF	1111 1111	17(S) 16(S) 15(S) 14(S) 13(S) 12(S) 11(S) 10(S)
0031	03	0000 0011	1F(N) 1E(N) 1D(N) 1C(N) 1B(N) 1A(N) 19(S) 18(S)
0032	FF	1111 1111	27(S) 26(S) 25(S) 24(S) 23(S) 22(S) 21(S) 20(S)
0033	01	0000 0001	2F(N) 2E(N) 2D(N) 2C(N) 2B(N) 2A(N) 29(N) 28(S)
0034	FE	1111 1110	37(S) 36(S) 35(S) 34(S) 33(S) 32(S) 31(S) 30(N)
0035	00	0000 0000	3F(N) 3E(N) 3D(N) 3C(N) 3B(N) 3A(N) 39(N) 38(N)

Figura 5
km_LoCapsable. ►

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0030		00 00	00 00 00 00	00

0036	00	0000 0000	47(N) 46(N) 45(N) 44(N) 43(N) 42(N) 41(N) 40(N)
0037	00	0000 0000	4F(N) 4E(N) 4D(N) 4C(N) 4B(N) 4A(N) 49(N) 48(N)
0038	00	0000 0000	57(N) 56(N) 55(N) 54(N) 53(N) 52(N) 51(N) 50(N)
0039	00	0000 0000	5F(N) 5E(N) 5D(N) 5C(N) 5B(N) 5A(N) 59(N) 58(N)
003A	00	0000 0000	67(N) 66(N) 65(N) 64(N) 63(N) 62(N) 61(N) 60(N)
003B	00	0000 0000	6F(N) 6E(N) 6D(N) 6C(N) 6B(N) 6A(N) 69(N) 68(N)
003C	00	0000 0000	77(N) 76(N) 75(N) 74(N) 73(N) 72(N) 71(N) 70(N)

Figura 6
km_HiCapsable.

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0030				FF BF FF
0040	EF FF EF FF	F7		

003D	FF	1111 1111	07(S) 06(S) 05(S) 04(S) 03(S) 02(S) 01(S) 00(S)
003E	BF	1011 1111	0F(S) 0E(N) 0D(S) 0C(S) 0B(S) 0A(S) 09(S) 08(S)
003F	FF	1111 1111	17(S) 16(S) 15(S) 14(S) 13(S) 12(S) 11(S) 10(S)
0040	EF	1110 1111	1F(S) 1E(S) 1D(S) 1C(N) 1B(S) 1A(S) 19(S) 18(S)
0041	FF	1111 1111	27(S) 26(S) 25(S) 24(S) 23(S) 22(S) 21(S) 20(S)
0042	EF	1110 1111	2F(S) 2E(S) 2D(S) 2C(N) 2B(S) 2A(S) 29(S) 28(S)
0043	FF	1111 1111	37(S) 36(S) 35(S) 34(S) 33(S) 32(S) 31(S) 30(S)
0044	F7	1111 0111	3F(S) 3E(S) 3D(S) 3C(S) 3B(N) 3A(S) 39(S) 38(S)

▲ *Figura 7*
km_LoRepeatable.

km_LoRepeatable e km_HiRepeatable

In figura 7 e figura 8 sono riportate le due tabelle contenenti le informazioni relative all'emissione continua del codice di scansione del singolo tasto. In ogni figura sono riportate due tabelle. La prima mostra la posizione del blocco della mappa di tastiera, la seconda mostra lo stesso blocco esploso in modo da riportarlo sia in formato esadecimale, che in formato binario. Questa tabella inoltre riporta i singoli codici di scansione con accanto, tra parentesi, **"S"** se la pressione continua del tasto produce l'emissione ripetuta del codice di scansione, **"N"** se tale codice viene comunque emesso una sola volta.

Ad esempio, il bit 0 del primo byte nella tabella relativa alla parte alta della mappa di tastiera, quella cioè riportata in figura 8, è **"1"**. Questo vuol dire che se si tiene premuto il tasto corrispondente al codice di scansione **0x40**, il tasto continuerà ad emettere sempre lo stesso carattere fintanto che non si alzerà il dito. Nel nostro caso il tasto in questione è la barra spaziatrice.

Viceversa, il bit 4 dello stesso byte vale **"0"**. Questo vuol dire che il tasto associatovi emetterà sempre e comunque una sola stringa di emissione, qualunque sia il tempo per il quale rimane premuto. Ed è bene che sia così, dato che stiamo parlando del tasto di *invio*, a cui corrisponde il codice di scansione **0x44**.

Da notare a questo riguardo, che i codici di scansione da **0x50** a **0x59**, possono emettere in continuazione. Dato che tali codici corrispondono ai tasti funzionali (*F1..F10*), fate attenzione se decidete di costruire una mappa di tastiera in cui tali tasti eseguono un comando (ad esempio **dir[invio]**). Sarebbe opportuno in tal caso azzerare il bit corrispondente.

km_LoKeyMapTypes e km_HiKeyMapTypes

In figura 9 e figura 10 sono riportate le due tabelle contenenti le informazioni relative al tipo dei tasti. In ogni figura è riportata una tabella ed una legenda. La tabella mostra la posizione del blocco nella mappa di tastiera, mentre la legenda fornisce la costante associata ad ognuno dei differenti tipi presenti in tabella. Ad esempio, alla posizione **0x72** (figura 9) è riportato il valore **0x27**. Dato che la tabella si riferisce alla mappa bassa della tastiera, se ne deduce che il tasto che emette il codice di scansione **0x26** è un tasto morto di tipo *vanilla*, che cioè supporta tutti e tre i modificatori. A questo tasto, nella tastiera italiana, corrisponde la lettera **"J"**.

km_LoKeyMap

In figura 11 è riportata la tabella contenente la mappa di tastiera vera e propria (sezione bassa). Si possono distinguere due tipi di parole: quelle che riportano le stringhe di emissione associate al singolo tasto secondo il *modello a quattro*, spiegato un paio di mesi fa, e quelle che contengono i puntatori a blocchi extra. Una analisi in dettaglio dell'intera mappa sarebbe decisamente lunga e del resto di scarsa utilità. Vediamo quindi solo un esempio per ogni tipo di tasto.

Per primo, vediamo ovviamente un *modello a quattro*. Prendiamo ad esempio quello alla posizione **0x0104**. Dato che si tratta della diciassettesima parola nella mappa "bassa", essa si riferirà al codice di scansione **0x10** (**17–> 0x11** con i codici che partono da **0x00**).

Vediamo subito dalla figura 9 (posizione **0x005C**) che si tratta di un tasto del tipo **KC_VANILLA**. La maggior parte di questi tasti sono lettere, quindi ci sono buone probabilità che sia un tasto alfabetico. In effetti, questa prima impressione è ulteriormente rafforzata dal fatto che dalla figura 5 e dalla figura 7 risulta che il tasto che ha codice di scansione **0x10** è sia influenzato dal blocco del maiuscolo, sia in grado di emettere in continuazione.

Dal modello a quattro si ricavano infine tutte le sequenze di emissione (ricordando la regola relativa all'effetto di *control* sui tasti *vanilla*), riportate in figura 13. Si tratta in effetti dal tasto identificato sulla tastiera dalla lettera **"Q"**. Le sequenze di emissione delle combinazioni contenenti *control* sono caratteri di controllo dell'*ECMA-94 Latin 1 International 8-bit character set*.

Vediamo adesso un tasto morto. Prendiamo la parola alla posizione **0x0144**. Si tratta di un puntatore ad un blocco extra il cui indirizzo è **0x0318**.

Per potere fare un'analisi corretta di questo blocco vediamo prima il tipo di tasto. La posizione del puntatore in tabella ci dice che stiamo parlando del

Figura 8
km_HiRepeatable.

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0040		47 F4 FF	03 00 00 00	
	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F

0046	47	0100 0111	47(N) 46(S) 45(N) 44(N) 43(N) 42(S) 41(S) 40(S)
0047	F4	1111 0100	4F(S) 4E(S) 4D(S) 4C(S) 4B(N) 4A(S) 49(N) 48(N)
0048	FF	1111 1111	57(S) 56(S) 55(S) 54(S) 53(S) 52(S) 51(S) 50(S)
0049	03	0011 0000	5F(N) 5E(N) 5D(S) 5C(S) 5B(N) 5A(N) 59(N) 58(N)
004A	00	0000 0000	67(N) 66(N) 65(N) 64(N) 63(N) 62(N) 61(N) 60(N)
004B	00	0000 0000	6F(N) 6E(N) 6D(N) 6C(N) 6B(N) 6A(N) 69(N) 68(N)
004C	00	0000 0000	77(N) 76(N) 75(N) 74(N) 73(N) 72(N) 71(N) 70(N)

Figura 9
km_LoKeyMapTypes.

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0040				07 03 07 03
0050	03 03 07 03	03 03 03 07	23 07 80 00	07 07 27 07
0060	07 27 27 27	27 07 07 07	80 05 00 00	27 07 07 27
0070	27 27 27 27	07 07 03 01	80 01 01 01	01 07 07 07
0080	07 07 27 07	03 03 07 80	00 01 01 05	
	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F

dove
00 è KC_NOQUAL
01 KCF_SHIFT
03 KCF_SHIFT + KCF_ALT
05 KCF_SHIFT + KCF_CONTROL
07 KC_VANILLA
23 KCF_DEAD + KCF_SHIFT + KCF_ALT
27 KCF_DEAD + KC_VANILLA
80 KCF_NOP

Figura 10
km_HiKeyMapTypes.

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
0080				22 00 41 00
0090	04 02 00 80	80 80 00 80	41 41 41 41	41 41 41 41
00A0	41 41 41 41	41 41 05 05	01 01 01 40	80 80 80 80
00B0	80 80 80 80	80 80 80 80	80 80 80 80	80 80 80 80
00C0	80 80 80 80			
	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F

dove
00 è KC_NOQUAL
01 KCF_SHIFT
02 KCF_ALT
04 KCF_CONTROL
05 KCF_SHIFT + KCF_CONTROL
22 KCF_DEAD + KCF_ALT
40 KCF_STRING
41 KCF_STRING + KCF_SHIFT
80 KCF_NOP

tasto con codice di scansione **0x20**. Dalla tabella in figura 9 vediamo che il tipo è **KCF_DEAD + KC_VANILLA**. Si tratta quindi di un tasto morto, ma ancora non sappiamo se muto o parlante. In effetti alcune combinazioni potrebbero essere mute, altre parlanti, altre ancora rappresentare un tasto normale. Il fatto che il tasto è *anche* di tipo *vanilla* ci dice inoltre che dobbiamo aspettarci ben otto descrittori nel blocco extra.

L'ordine dei descrittori è quello già riportato in figura 2 della 42ª puntata, e che per comodità riportiamo di nuovo in figura 14 dove è anche riportata l'analisi completa del blocco extra. Questo è composto da due parti. La prima contiene otto descrittori da due byte l'uno, la seconda contiene la matrice delle sequenze di emissione. Vediamo subito che il primo byte di ogni descrittore è

La scheda tecnica: Inside 2.0

Con questa puntata vedremo le ultime schede tecniche relative alle funzioni della **gadtools.library**. Si tratta di ben dodici funzioni, ma dato che siamo arrivati all'ultima puntata di *Programmare in C su Amiga* dovevamo in qualche modo concludere la presentazione della libreria.

FreeMenus

Libera la memoria allocata dalla **CreateMenusA()**.

prototipo

```
VOID FreeMenus  // Non ritorna alcun risultato
(
  struct Menu *menu  // Puntatore alla struttura menù (od alla prima
                     // voce) fornito dalla CreateMenusA()
) ;
```

FreeVisualInfo

Libera tutte le risorse ottenute tramite la **GetVisualInfoA()**.

prototipo

```
VOID FreeVisualInfo  // Non ritorna alcun risultato
(
  APTR vi  // Puntatore fornito dalla GetVisualInfoA()
) ;
```

Questa funzione va chiamata solo dopo aver chiuso la finestra (od il quadro) ma prima di chiudere lo schermo, sia che si usi **CloseScreen()** o **unlockPubScreen()**.

GetVisualInfoA

Serve ad ottenere tutta una serie di informazioni necessarie ai controlli di tipo **GadTools** per una corretta visualizzazione degli stessi.

prototipo

```
APTR GetVisualInfoA  // Ritorna il puntatore ad un blocco dati privato
(
  struct Screen  *screen , // Puntatore allo schermo su cui si opera
  struct TagItem *taglist  // Puntatore ad una lista di "tag"
) ;
```

Le informazioni ottenute assicurano una rete ottimale dei controlli qualunque siano le caratteristiche dello schermo su cui si apriranno le finestre. Questa funzione, infatti, va chiamata dopo aver aperto uno schermo, o acquisito uno schermo pubblico, ma prima di creare nuovi controlli o nuove finestre su di esso.
Il blocco di cui viene fornito il puntatore contiene infatti una serie di dati che vanno passati in ingresso quando si crea un nuovo controllo od un nuovo menu. Si tratta di un blocco *privato* del sistema che non va analizzato dal programmatore. Anzi, non cercate di capirne la struttura, perché questa potrebbe variare nelle nuove versioni del sistema operativo.

GetVisualInfo

È la versione a parametri variabili della **GetVisualInfoA()**.

prototipo

```
APTR GetVisualInfo  // Ritorna il puntatore ad un blocco dati privato
(
  struct Screen  *screen   , // Puntatore allo schermo su cui si opera
  (tagtype)       firsttag , // Primo tag
  ...
) ;
```

GT_BeginRefresh

Inizia la sequenza di restauro per i controlli.

prototipo

```
VOID GT_BeginRefresh  // Non ritorna alcun risultato
(
  struct Window *win  // Puntatore alla finestra dalla quale è stato
                      // ricevuto l'evento IDCMP_REFRESHWINDOW
) ;
```

Questa funzione chiama la funzione di Intuition **BeginRefresh()** in un modo particolarmente congeniale ai controlli di tipo **GetTools**, i quali possono iniziare così tutte le operazioni necessarie al loro completo restauro.

La sequenza minimale da supportare in caso di richiesta di restauro è la seguente:

```
case IDCMP_REFRESHWINDOW:   GT_BeginRefresh(win) ;
                            GT_EndRefresh(win, TRUE) ;
                            break ;
```

Nota: a causa della loro natura, l'utilizzo dei controlli di tipo **GadTools** preclude la possibilità di usare **WFLG_NOCARERE-FRESH**.

GT_EndRefresh

Termina la sequenza di restauro per i controlli.

0x00 o **0x01**. Ne segue che le varie combinazioni tra i modificatori e questo tasto hanno le caratteristiche o di un carattere normale, oppure di un tasto morto parlante. Nel primo caso il byte che segue dà direttamente il carattere da emettere, nel secondo dà la distanza tra il descrittore ed un elemento della matrice delle sequenze di emissione. Ad esempio, il primo descrittore punta alla **"a"** minuscola, mentre il terzo contiene direttamente il carattere da emettere **"æ"**.

Anche in questo caso le combinazioni con *control* emettono caratteri di controllo, come ad esempio **SOH**.

Da notare che la matrice delle sequenze di emissione relativa alla pressione del tasto da solo è formata da sei byte, dal che si deduce che la tastiera italiana ha solo cinque tasti morti muti

```
prototipo
VOID GT_EndRefresh  // Non ritorna alcun risultato
(
  struct Window *win     , // Puntatore alla finestra dalla quale è stato
                           // ricevuto l'evento IDCMP_REFRESHWINDOW
  BOOL           complete // TRUE se il restauro è stato effettuato
) ;
```

Questa funzione chiama la funzione di Intuition **EndRefresh()** in un modo particolarmente congeniale ai controlli di tipo **GadTools**, i quali possono completare così tutte le operazioni necessarie al restauro iniziato tramite la **GT_BeginRefresh().**

GT_FilterIMsg

Serve a filtrare un messaggio di Intuition.

```
prototipo
struct IntuiMessage *GT_FilterIMsg  // Messaggio filtrato
(
  struct IntuiMessage *imsg  // Messaggio originale (da filtrare)
) ;
```

Questa funzione prende il messaggio passatogli dal programma e lo esamina. Se esso può essere gestito completamente da un controllo di tipo **GadTools**, allora ritorna **NULL**, altrimenti restituisce il messaggio, eventualmente contenente ulteriori informazioni, per l'analisi diretta da parte del programma. Da tener presente che non va effettuato alcun confronto tra il messaggio originale e quello filtrato. Una interpretazione basata su tale confronto potrebbe infatti essere fuorviante.

Una volta terminata l'analisi del messaggio filtrato, si può usare la **GT_PostFilterIMsg()** per ricostruire quello originale.

Nota: si tratta di una funzione che ha senso utilizzare solo in casi molto particolari.

GT_GetIMsg

Riceve un messaggio da Intuition e lo fa pre-processare direttamente dal controllo interessato.

```
prototipo
struct IntuiMessage *GT_GetIMsg  // Messaggio filtrato
(
  struct MsgPort *intuiport  // Porta di comunicazione con Intuition
) ;
```

Va usata al posto della funzione di *exec* **GetMsg()**.

Riceve un messaggio di Intuition, lo fa analizzare dal controllo di tipo **GadTools** interessato, e poi restituisce il messaggio modificato al programma chiamante, liberando così il programmatore dalla gestione degli eventi che non coinvolgono una logica particolare.

Se il messaggio può essere gestito completamente dal controllo, allora la funzione ritorna **NULL**, altrimenti restituisce il messaggio, eventualmente contenente ulteriori informazioni, per l'analisi diretta da parte del programma.

Nota: per rispondere al messaggio, utilizzare la **GT_ReplyIMsg()**, e non la **ReplyMsg()**.

GT_PostFilterIMsg

Restituisce il messaggio originale dopo che questi è stato filtrato dalla **GT_FilterIMsg()**.

```
prototipo
struct IntuiMessage *GT_PostFilterIMsg  // Messaggio originale
(
  struct IntuiMessage *modimsg  // Messaggio filtrato
) ;
```

Anche in questo caso non va effettuato alcun confronto tra il messaggio originale e quello filtrato. Una interpretazione basata su tale confronto potrebbe infatti essere fuorviante. Tutto ciò che si può fare con il messaggio ritornato è accodarlo ancora, o rispondere. Tra l'altro, dato che in questo caso è stata usata la **GetMsg()** per acquisire il messaggio, è necessario utilizzare la **ReplyMsg()**.

GT_RefreshWindow

Ripristina tutti i controlli di tipo *GadTools*.

```
prototipo
VOID GT_RefreshWindow  // Non ritorna alcun valore
(
  struct Window    *win , // Finestra a cui appartengono i controlli
  struct Requester *req   // Quadro, se di tale si tratta, o NULL
) ;
```

Da chiamare subito dopo aver aperto una finestra, oppure dopo aver aggiunto dei controlli con **AddGList()** ed aver chiamato **RefreshGList**. Per il momento il secondo parametro va sempre impostato a **NULL**, dato che i controlli di tipo *GadTools* non sono ancora utilizzabili per i quadri.

▶

GT_ReplyIMsg

Risponde ad un messaggio acquisito con **GT_GetIMsg()**.

prototipo

```
VOID GT_ReplyIMsg  // Non ritorna alcun valore
(
  struct IntuiMessage *imsg  // Messaggio a cui si risponde
) ;
```

Va usata al posto della funzione di *exec* **ReplyMsg()**, quando si usano controlli di tipo *GadTools*. Non chiamare la **CloseWindow()** senza aver prima risposto a tutti i messagi ancora in coda con questa funzione.

GT_SetGadgetAttrsA

Serve a modificare le caratteristiche di un controllo di tipo *GadTools*.

prototipo

```
VOID GT_SetGadgetAttrsA  // Non ritorna alcun risultato
(
  struct Gadget    *gad    , // Controllo da modificare
  struct Window    *win    , // Finestra che lo contiene
  struct Requester *req    , // Quadro che lo contiene, o NULL
  struct TagItem   *taglist  // Lista di "tag"
) ;
```

Qui di seguito è riportata la scheda relativa ai vari *tag* validi per questa funzione.

Controllo: BUTTON_KIND	Pulsante a pressione
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default).

Controllo: CHECKBOX_KIND	Casella di spunta
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default).
GTCB_Checked BOOL	Stato iniziale della casella di spunta. Il default è FALSE.

Controllo: CYCLE_KIND	Selezione ciclica
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default)
GTCY_Active UWORD	Numero d'ordine (a partire da zero) della scelta iniziale (default = 0).
GTCY_Labels STRPTR *	Puntatore ad un vettore di stringhe corrispondenti alle possibile scelte. L'ultima è nulla.

Controllo: INTEGER_KIND	Campo intero
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default).
GTIN_Number LONG	Valore iniziale del campo (default = 0).

Controllo: LISTVIEW_KIND	Elenco scorrevole
Tag / Tipo	Descrizione
GTLV_Labels struct List *	Lista di voci. Il testo da visualizzare nell' elenco è nei campi "ln_Name". Usa 0 per "staccare" la lista dallo schermo, prima di modificarne il contenuto. Poi riattaccala con {GTLV_Labels, list}.
GTLV_Selected UWORD	Numero d'ordine della voce selezionata, o 0 se nessuna voce è selezionata (default = 0).
GTLV_Top UWORD	Prima voce visibile nell'elenco (default = 0).

Controllo: MX_KIND	Gruppo di bottoni mutualmente esclusivi
Tag / Tipo	Descrizione
GTCY_Active UWORD	Numero d'ordine (a partire da zero) del bottone inizialmente selezionato (default = 0).

Controllo: NUMBER_KIND	Campo numerico (sola lettura)
Tag / Tipo	Descrizione
GTNM_Number LONG	Intero (long) con segno da visualizzare in sola lettura (default = 0).

Controllo: PALETTE_KIND	Tavolozza dei colori
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default).
GTPA_Color UBYTE	Colore inizialmente selezionato (default = 1).

Controllo: SCROLLER_KIND	Barra di scorrimento
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default).
GTSC_Top WORD	Posizione del primo elemento visibile nella lista gestita dalla barra (default = 0).
GTSC_Total WORD	Numero totale di elementi della lista gestita dalla barra (default = 0).
GTSC_Visible WORD	Numero totale di elementi visibili della lista gestita dalla barra (default = 2).

Controllo: SLIDER_KIND	Indicatore di livello
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default).
GTSL_Level WORD	Livello attuale in cui si trova il cursore dell'indicatore (default = 0).
GTSL_Max WORD	Livello massimo a cui può arrivare il cursore dell'indicatore (default = 15).
GTSL_Min WORD	Livello minimo a cui può arrivare il cursore dell'indicatore (default = 0).

Controllo: STRING_KIND	Campo caratteri
Tag / Tipo	Descrizione
GA_Disabled BOOL	TRUE disabilita il controllo, altrimenti FALSE (default).
GTST_String STRPTR	Valore iniziale del campo, o NULL (default) se il campo deve essere inizialmente vuoto.

Controllo: TEXT_KIND	Testo statico (sola lettura)
Tag / Tipo	Descrizione
GTTX_Text BOOL	Puntatore ad una stringa di caratteri da visualizzare in sola lettura, o NULL (default).

GT_SetGadgetAttrs

Serve a modificare le caratteristiche di un controllo di tipo *GadTools*. Versione a parametri variabili della **GT_SetGadgetAttrsA()**.

prototipo

```
VOID GT_SetGadgetAttrs  // Non ritorna alcun risultato
(
  struct Gadget    *gad      , // Controllo da modificare
  struct Window    *win      , // Finestra che lo contiene
  struct Requester *req      , // Quadro che lo contiene, o NULL
  (tagtype)         firsttag , // Primo tag
  ...
) ;
```

LayoutMenuItemsA

Posiziona tutte le voci dei menu.

prototipo

```
BOOL LayoutMenuItemsA  // Torna FALSE se non ha potuto aprite TextAttr
(
  struct MenuItem  *menuitem , // Lista a catena di voci
  APTR              vi       , // Puntatore fornito da GetVisualInfoA()
  struct TagItem   *taglist    // Lista di "tag"
) ;
```

Questa funzione va usata nel caso sia stata creata una struttura di voci relativa ad un singolo menu con la **CreateMenusA()**.

Qui di seguito è riportata la scheda relativa ai vari *tag* validi per questa funzione.

Tag / Tipo	Descrizione
GTMN_TextAttr struct TextAttr *	Attributi del testo da utilizzare per i menù e per le voci. Il default sono quelli dello schermo.
GTMN_Menu struct Menu *	Puntatore ad un menù la cui prima voce è quella fornita alla funzione come parametro.

LayoutMenuItems

Posiziona tutte le voci dei menu. Versione a parametri variabili della **LayoutMenuItemsA()**.

prototipo

```
BOOL LayoutMenuItems  // Torna FALSE se non ha potuto aprite TextAttr
(
  struct MenuItem  *menuitem , // Lista a catena di voci
  APTR              vi       , // Puntatore fornito da GetVisualInfoA()
  (tagtype)         firsttag , // Primo tag
  ...
) ;
```

LayoutMenusA

Posiziona tutti i menu e le loro voci.

prototipo

```
BOOL LayoutMenusA  // Torna FALSE se non ha potuto aprite TextAttr
(
  struct Menu      *menu     , // Struttura di menù completa
  APTR              vi       , // Puntatore fornito da GetVisualInfoA()
  struct TagItem   *taglist    // Lista di "tag"
) ;
```

Questa funzione va usata nel caso sia stata creata una struttura di menu con la **CreateMenusA()**.

Qui di seguito è riportata la scheda relativa ai vari *tag* validi per questa funzione.

Tag / Tipo	Descrizione
GTMN_TextAttr struct TextAttr *	Attributi del testo da utilizzare per i menù e per le voci. Il default sono quelli dello schermo.

LayoutMenus

Posiziona tutti i menu e le loro voci. Versione a parametri variabili della **LayoutMenusA()**.

prototipo

```
BOOL LayoutMenus  // Torna FALSE se non ha potuto aprite TextAttr
(
  struct Menu      *menu     , // Lista a catena di voci
  APTR              vi       , // Puntatore fornito da GetVisualInfoA()
  (tagtype)         firsttag , // Primo tag
  ...
) ;
```

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
00C0		7E 60 7E 60	21 B9 21 31	40 B2 22 32
00D0	23 B3 A3 33	B0 A2 24 34	25 BC 25 35	5E BD 26 36
00E0	26 BE 2F 37	2A B7 28 38	28 AB 29 39	29 BB 3D 30
00F0	5F 2D 3F 27	00 00 02 A4	7C 5C 7C 5C	00 00 00 00
0100	00 00 00 30	C5 E5 51 71	B0 B0 57 77	00 00 03 34
0110	AE AE 52 72	DE FE 54 74	00 00 02 FC	00 00 03 A4
0120	00 00 03 50	00 00 03 88	B6 B6 50 70	7B 5B E9 E8
0130	7D 5D 2A 2B	00 00 00 00	00 00 7C 31	00 00 00 32
0140	00 00 00 33	00 00 03 18	A7 DF 53 73	D0 F0 44 64
0150	00 00 02 AC	00 00 02 BC	00 00 02 CC	00 00 02 DC
0160	00 00 02 EC	A3 A3 4C 6C	3A 3B 40 F2	22 27 23 E0
0170	00 00 A7 F9	00 00 00 00	00 00 7B 34	00 00 7D 35
0180	00 00 7E 36	00 00 3E 3C	AC B1 5A 7A	F7 D7 5B 78
0190	C7 E7 43 63	AA AA 56 76	BA BA 42 62	00 00 03 6C
01A0	BF B8 4D 6D	3C 2C 3B 2C	3E 2E 3A 2E	3F 2F 5F 2D
01B0	00 00 00 00	00 00 00 2E	00 00 2F 37	00 00 B0 38
01C0	1C 1C 5C 39			
	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F

Blocchi extra

00F4	00 00 02 A4
010C	00 00 03 34
0118	00 00 02 FC
011C	00 00 03 A4
0120	00 00 03 50
0124	00 00 03 88
0144	00 00 03 18

0150	00 00 02 AC
0154	00 00 02 BC
0158	00 00 02 CC
015C	00 00 02 DC
0160	00 00 02 EC
019C	00 00 03 6C

Figura 11 - km_LoKeyMap.

	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F
01C0		00 00 03 C0	00 00 00 08	00 00 03 CA
01D0	00 00 00 0D	00 00 0A 0D	00 00 9B 1B	00 00 00 7F
01E0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 2D
01F0	00 00 00 00	00 00 03 D1	00 00 03 D9	00 00 03 E1
0200	00 00 03 EA	00 00 03 F3	00 00 03 FE	00 00 04 09
0210	00 00 04 14	00 00 04 1F	00 00 04 2A	00 00 04 35
0220	00 00 04 40	00 00 04 4B	00 00 04 56	1B 1B 7B 5B
0230	1D 1D 7D 5D	00 00 2F 2F	00 00 2A 2A	00 00 2B 2B
0240	00 00 04 61	00 00 00 00	00 00 00 00	00 00 00 00
0250	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0260	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0270	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0280	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0290	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
02A0	00 00 00 00			
	00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F

Blocchi extra

01C4	00 00 03 C0
01CC	00 00 03 CA
01F4	00 00 03 D1
01F8	00 00 03 D9
01FC	00 00 03 E1
0200	00 00 03 EA
0204	00 00 03 F3
0208	00 00 03 FE
020C	00 00 04 09

0210	00 00 04 14
0214	00 00 04 1F
0218	00 00 04 2A
021C	00 00 04 35
0220	00 00 04 40
0224	00 00 04 4B
0228	00 00 04 56
0240	00 00 04 61

Figura 12 - km_HiKeyMap.

differenti. In effetti potrebbero essere di più, ma in questo caso quelli in eccesso avrebbero lo stesso effetto di uno dei cinque base.

Vediamo infine un tasto morto muto. Prendiamo ad esempio la trentasettesima parola nella mappa bassa, quella cioè alla posizione **0x0154**, che punta al blocco extra all'indirizzo **02BC**. L'analisi è riportata in figura 15. Il tasto è quello che sulla tastiera è contrassegnato dalla lettera **"G"**. In questo caso abbiamo solo gli otto descrittori. Vediamo subito che le due combinazioni mute sono quelle che contengono *alt*. In questo caso il secondo byte indica l'indice dell'elemento nella matrice delle sequenze di emissione di un tasto morto parlante che deve essere emesso.

Se ad esempio avessimo premuto prima **ALT-G** e poi **A**, avremmo ottenuto il carattere alla posizione **0x032A** in figura 14, cioè **0xE0**, ovverosia **"à"**.

km_HiKeyMap

In figura 12 è riportata la tabella contenente la mappa di tastiera vera e propria (sezione alta).

Di questa tabella vediamo solo un tasto di tipo stringa.

Prendiamo ad esempio quello corrispondente alla quattordicesima parola, che punta cioè al blocco extra all'indirizzo **0x03D9**. Il codice di scansione corrispondente è **0x4D**. Dalle varie tabelle relative alla mappa "alta", vediamo subito che questo tasto non è influenzato dal blocco del maiuscolo, ma che può emettere in continuazione. Il tipo è **0x41** e cioè **KCF_STRING + KCF_SHIFT**. Ci aspettiamo quindi due descrittori.

L'analisi del blocco extra è riportata in figura 16.

Come si vede in figura, possono essere emesse due stringhe. La prima è **<CSI>B**, che corrisponde allo spostamento del cursore verso il basso (freccia in giù). La seconda, relativa all'uso in contemporanea dello *shift*, è **<CSI>S**, che corrisponde allo scorrimento verso l'alto del testo di una riga (e cioè alla visualizzazione della successiva riga in basso).

modello a quattro	C5 E5 51 71

Combinazione premuta	Valore	Carattere
tasto da solo	71	q
shift + tasto	51	Q
alt + tasto	E5	å
shift + alt + tasto	C5	Å
control + tasto	11	DC1
control + shift + tasto	01	SOH
control + alt + tasto	95	MW
control + shift + alt + tasto	85	NEL

Figura 13
Modello a quattro.

Conclusione

E proprio di conclusione si tratta, dato che, dopo esattamente quattro anni da

```
          00 01 02 03   04 05 06 07   08 09 0A 0B   0C 0D 0E 0F
0310
0320    00 01 00 01   00 81 00 81   01 10 01 16   00 E6 00 C6
0330    C0 C2 C3 C4                 61 E1 E0 E2   E3 E4 41 C1
          00 01 02 03   04 05 06 07   08 09 0A 0B   0C 0D 0E 0F
```

N	Tipo	Valore	Emette	Combinazione corrispondente
1	01 parlante	10 punta a 61	a	tasto
2	01 parlante	16 punta a 41	A	shift + tasto
3	00 normale	E6 carattere	æ	alt + tasto
4	00 normale	C6 carattere	Æ	shift + alt + tasto
5	00 normale	01 carattere	SOH	control + tasto
6	00 normale	01 carattere	SOH	control + shift + tasto
7	00 normale	81 carattere	**	control + alt + tasto
8	00 normale	81 carattere	**	control + shift + alt + tasto

Figura 14 - Tasto morto parlante.

```
          00 01 02 03   04 05 06 07   08 09 0A 0B   0C 0D 0E 0F
02B0                                                00 67 00 47
02C0    08 02 08 02   00 07 00 07   00 87 00 87
          00 01 02 03   04 05 06 07   08 09 0A 0B   0C 0D 0E 0F
```

N	Tipo	Valore	Emette	Combinazione corrispondente
1	00 normale	67 carattere	g	tasto
2	00 normale	47 carattere	G	shift + tasto
3	08 muto	02 indice	2° (*)	alt + tasto
4	08 muto	02 indice	2° (*)	shift + alt + tasto
5	00 normale	07 carattere	BEL	control + tasto
6	00 normale	07 carattere	BEL	control + shift + tasto
7	00 normale	87 carattere	ESA	control + alt + tasto
8	00 normale	87 carattere	ESA	control + shift + alt + tasto

(*) Emette il secondo elemento nella matrice delle sequenze di emissione di un tasto morto parlante.

Figura 15 - Tasto morto muto.

```
          00 01 02 03   04 05 06 07   08 09 0A 0B   0C 0D 0E 0F
03D0                                    02 04 02    06 9B 42 9B
03E0    53
          00 01 02 03   04 05 06 07   08 09 0A 0B   0C 0D 0E 0F
```

N	Lunghezza	Posizione	Stringa	Combinazione corrispondente
1	02	04	9B 42 <CSI> B	tasto
2	02	06	9B 53 <CSI> S	shift + tasto

Figura 16 - Tasto stringa.

quando la prima puntata è stata pubblicata su *MCmicrocomputer*, è stato deciso di chiudere questa rubrica, ormai effettivamente protrattasi per troppo tempo. Le esigenze di rinnovamento tipiche di un mercato in continua evoluzione, e la necessità di fornire ai lettori sempre nuovi punti di interesse, richiedono una struttura più flessibile e dinamica. Si supera così il vecchio concetto del lettore affezionato, disposto a seguire (e sorbirsi) ben 44 puntate di una stessa rubrica — complimenti a chi ci è riuscito! — per allargare la propria offerta anche al lettore occasionale, o a chi non compra regolarmente la rivista.

Per questo abbiamo deciso di abbandonare il vecchio modello a puntate, utilizzato fino ad oggi per questa rubrica, e di sviluppare una nuova serie di rubriche estremamente articolate che, sviluppandosi al massimo su due o tre puntate, vanno a coprire argomenti differenziati e spesso molto diversi fra loro.

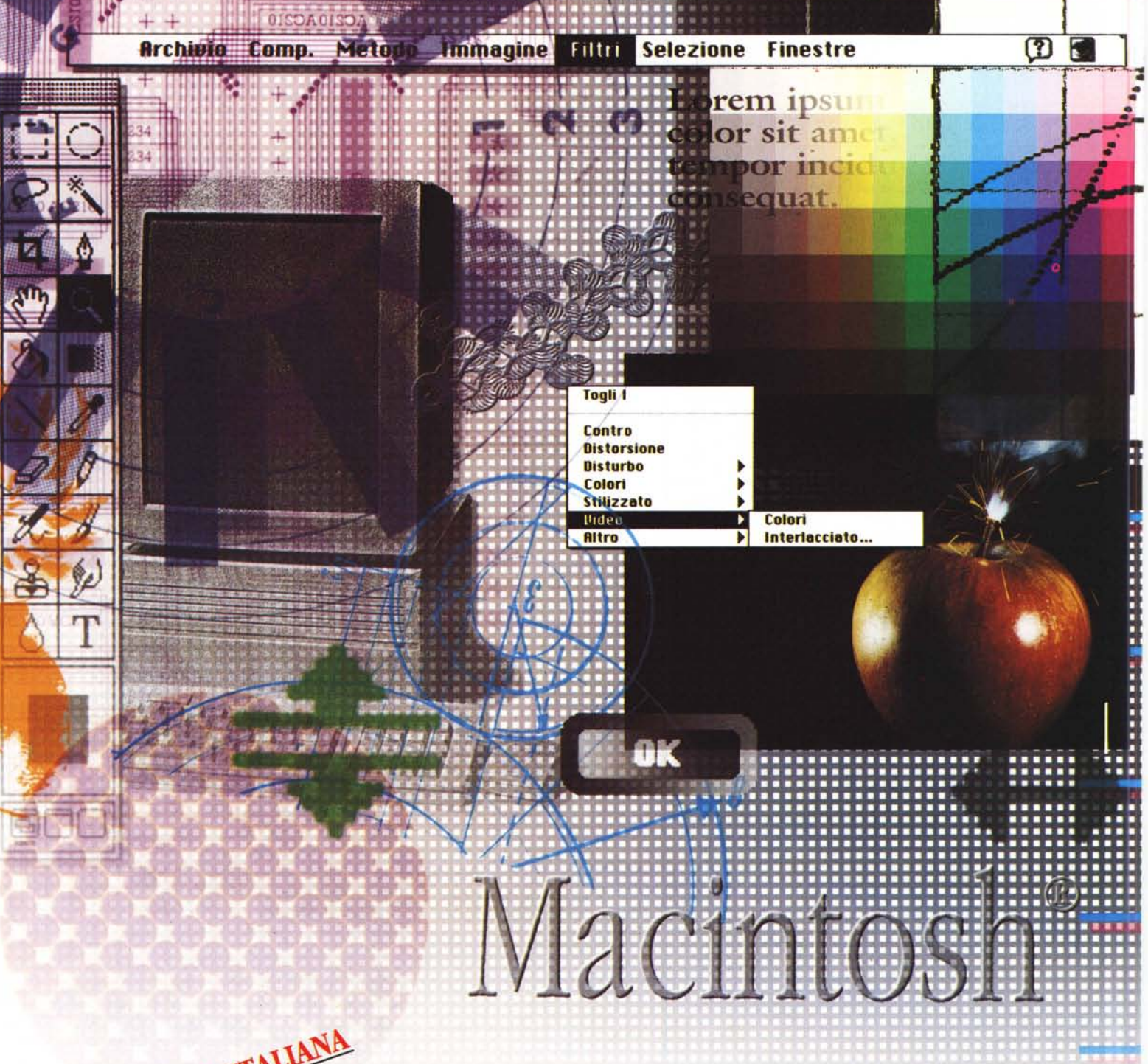
Naturalmente parleremo sempre di Amiga, e nella maggior parte dei casi si tratterà comunque di articoli tecnici, orientati a chi non si accontenta di usare i programmi scritti dagli altri, ma desidera creare qualcosa di suo. Non parleremo tuttavia solo di C, e neanche solo di programmazione. Non ci limiteremo neanche al solo sistema operativo, ma analizzeremo anche vari prodotti che forniscono linguaggi proprietari o macro di programmazione. Cercheremo insomma di vedere le cose da un punto di vista diverso, non limitandoci a provare e presentare un prodotto (altri lo fanno già brillantemente sulle pagine di questa rivista), ma scavando un po' più a fondo e fornendo spunti un po' particolari a chi è interessato a capire il funzionamento delle cose. Parleremo quindi di *BOOPSI*, di *AmigaVision*, dell'*Amiga OS 2.1*, di *Commodities*, di librerie *ARexx* non standard, e di molto altro ancora.

Avremo inoltre qualche articolo di interludio dedicato alle novità del mondo Amiga, alle tecniche di analisi e di disegno, a quelle di *test* dei programmi.

In definitiva *È morto il re, viva il re*. Mi spiace solo che questo cambiamento sia avvenuto un po' a sorpresa, deludendo forse i miei quattro lettori più affezionati. D'altra parte *lo spettacolo deve continuare*, no?

MC

Dario de Judicibus è raggiungibile tramite MC-Link alla casella MC2120.



1ª EDIZIONE ITALIANA

MACWORLD EXPOSITION

Milano, 14-16 Maggio 1992

Mostra Convegno del Mercato dei Sistemi Macintosh®

SPAZIO MILANO*NORD*
Via Pompeo Mariani, 2 - Milano
(M1 Precotto - M2 Cimiano - Bus 44)

Orario: 9.00-18.00

Segreteria Generale
"MACWORLD EXPO":
Via Domenichino, 11 - 20149 Milano
(C.P. 15117 - 20150 Milano)
Tel. 02/4815541 - Fax 02/4980330
Telex 313627

È un'iniziativa:
WORLD EXPO CORPORATION • ASSOEXPO

— Macintosh è un marchio registrato di Apple Computer —